C++语言程序设计

第二章 C++简单程序设计

中国科大　　黄章进

本章主要内容

- 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
- 自定义数据类型
- 深度探索

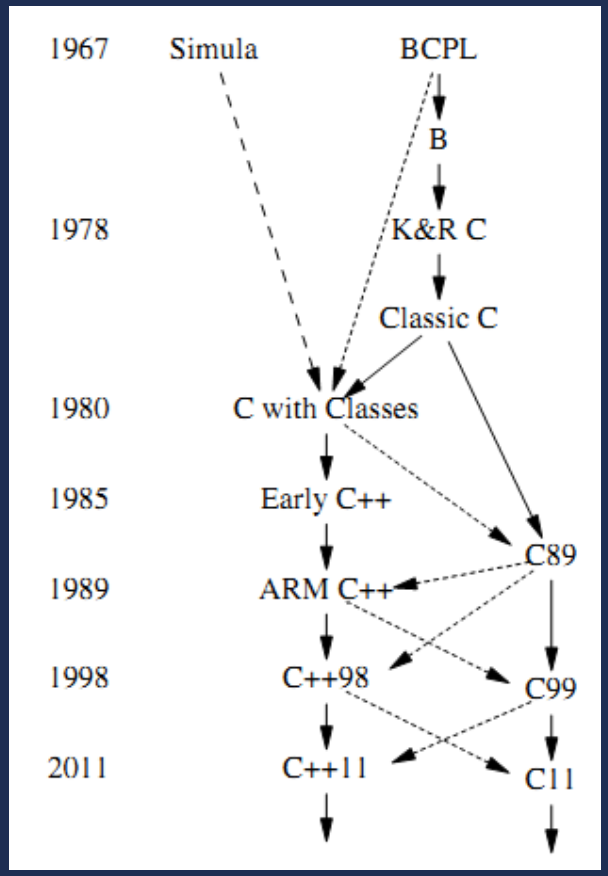
C++语言的产生

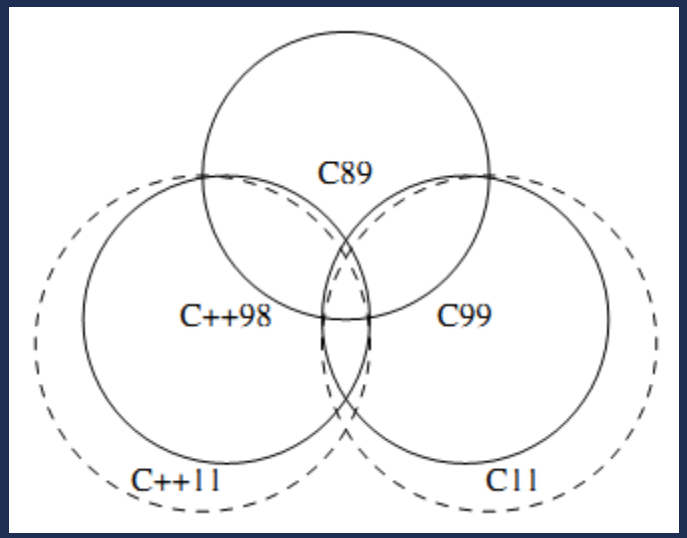
C++语言概述

- C++是从C语言发展演变而来的，首先是一个更好的C
- 引入了类的机制，最初的C++被称为“带类的C”
- 1983年正式取名为C++
- 从1989年开始C++语言的标准化工作
- 于1994年制定了ANSI C++标准草案
- 于1998年11月被国际标准化组织（ISO）批准为国际标准，第一版的C++98(C++03)
- 于2011年8月发布了第二版的ISO C++标准：C++11

C++和C是兄弟

C++语言概述





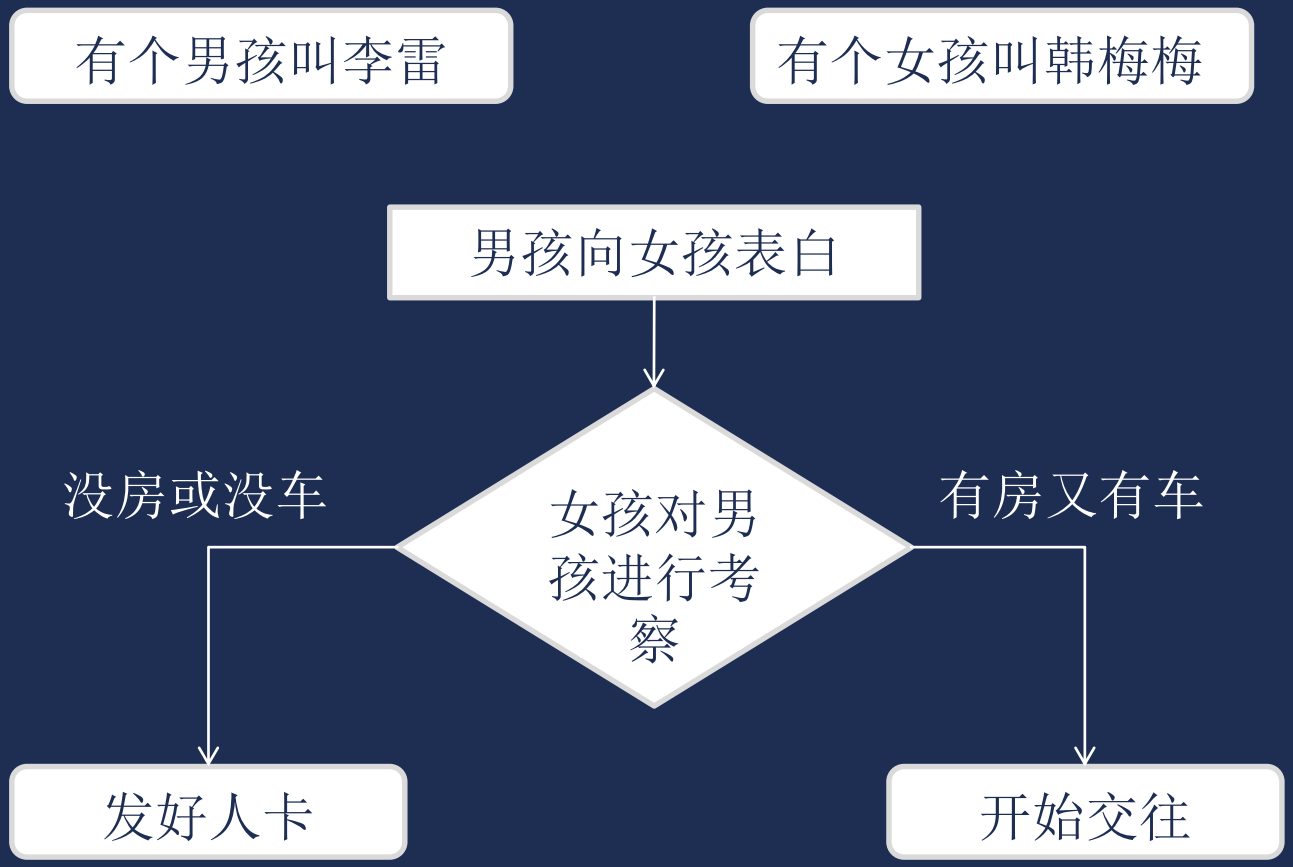
C++的特点

C++语言概述

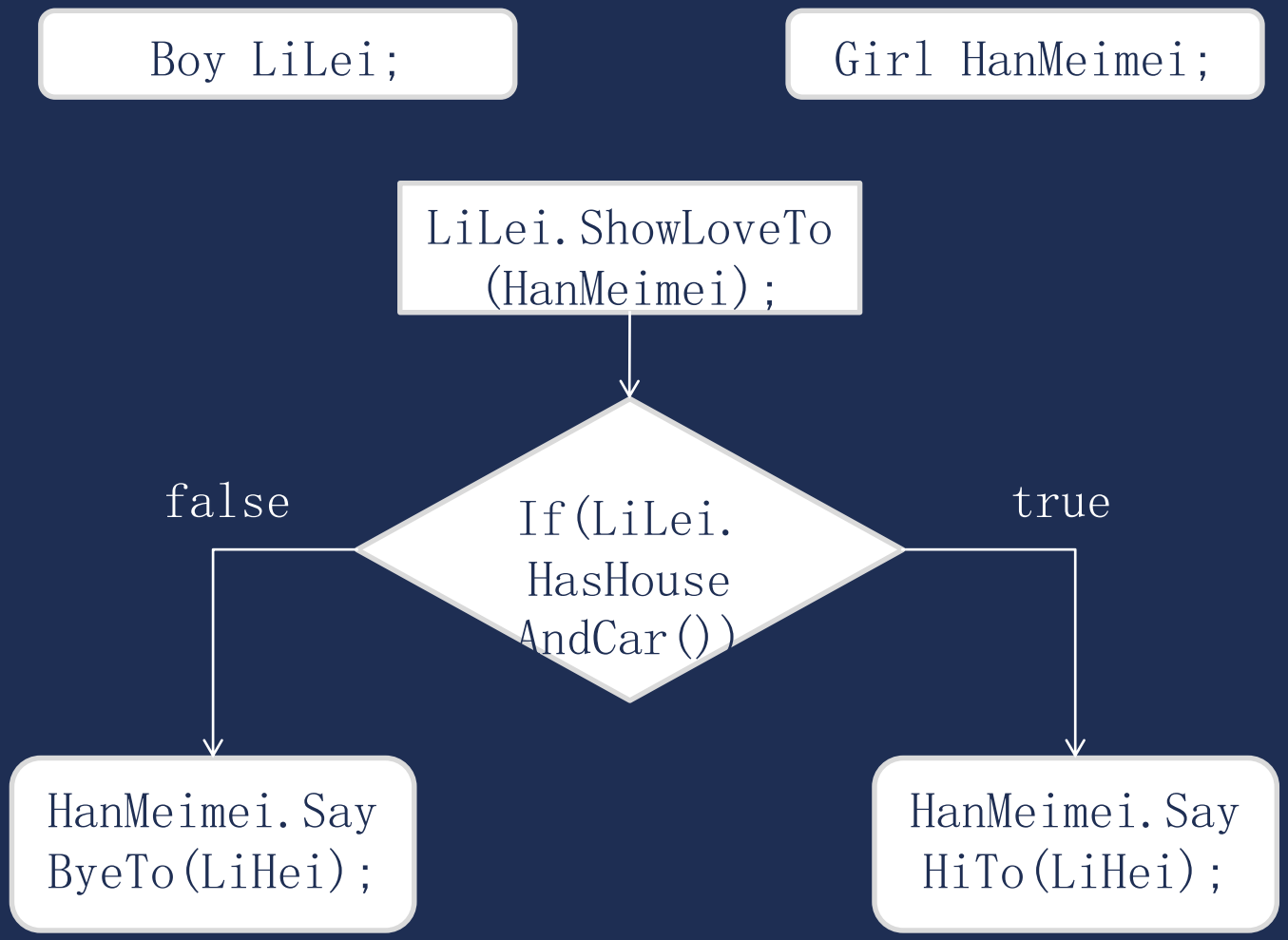
- 兼容C
  - 它保持了C的简洁、高效和接近汇编语言等特点
  - 对C的类型系统进行了改革和扩充
  - C++也支持面向过程的程序设计，不是一个纯正的面向对象的语言
- 支持面向对象的方法

李雷和韩梅梅

C++语言概述



李雷和韩梅梅


Boy LiLei;
Girl HanMeimei;
LiLei.ShowLoveTo (HanMeimei);
If(LiLei. HasHouse AndCar())
false
true
HanMeimei.Say ByeTo(LiHei);
HanMeimei.Say HiTo(LiHei);

自然语言翻译成C++语言

C++语言概述

```
//有个男孩叫李雷
Boy  LiLei;
//有个女孩叫韩梅梅
Girl  HanMeimei;

//男孩向女孩示爱
LiLei.ShowLoveTo(HanMeimei);
//女孩对男孩进行考察
if (LiLei.HasHouseAndCar() )
{
    //如果有房又有车,女孩与男孩交往
    HanMeimei.SayHiTo(LiLei);
}
else
{
    //如果没有，则与男孩拜拜
    HanMeimei.SayByeTo(LiLei);
}
```

C++程序实例—例2-1

C++语言概述

```
//2_1.cpp
#include <iostream>
using namespace std;
int main() {
  cout << "Hello!" << endl;
  cout << "Welcome to c++!" << endl;
  return 0;
}
```

运行结果：

```
Hello!
Welcome to c++!
```

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：

!	#	%	^	&	*	_
+	=	-	~	<	>	/
\	'	"	;	.	,	:
?	(	)	[	]	{	}

词法记号

C++语言概述

- 关键字　C++预定义的单词
- 标识符　程序员声明的单词，它命名程序正文中的一些实体
- 文字　在程序中直接使用符号表示的数据
- 运算符　用于实现各种运算的符号
- 分隔符 ()　{}　,　:　;
  用于分隔各个词法记号或程序正文
- 空白符　空格、制表符（TAB键产生的字符）、垂直制表符、换行符、回车符和注释的总称

C++注释方法：1./* 注释。。。 */
2. // 注释。。。

标识符的构成规则

- 以大写字母、小写字母或下划线(_)开始。
- 可以由以大写字母、小写字母、下划线(_)或数字0～9组成。
- 大写字母和小写字母代表不同的标识符。
- 不能是C++关键字

  – Boy, boy, Draw_line, _No1　　No.1, 1st

数据类型

- 基本类型
  - void
  - 算术类型
    - 整型
      - 布尔型：bool
      - 字符型：char, w_char_t, char16_t, char32_t
      - 整型：short, int, long, long long
    - 浮点型
      - float, double, long double
- 复合类型
- 自定义类型

基本数据类型和表达式

数据类型
——常量与变量

```
#include <iostream>
using namespace std;
int main() {
  const int PRICE = 30;
  int num, total;
  float v, r, h;
  num = 10;
  total = num * PRICE;
  cout << total << endl;
  r = 2.5;
  h = 3.2;
  v = 3.14159 * r * r * h;
  cout << v << endl;
  return 0;
}
```

变量先声明后使用

符号常量

常量

变量

数据类型
——整型数据

基本数据类型和表达式

整型变量声明

```
#include <iostream>
using namespace std;
int main() {
  const int PRICE = 30;
  int num, total;
  float v, r, h;
  num = 10;
  total = num * PRICE;
  cout << total << endl;
  r = 2.5;
  h = 3.2;
  v = 3.14159 * r * r
  cout << v << endl;
  return 0;
}
```

整型变量

整型常量

short	
unsigned short	2字节
int	
unsigned [int]	4字节
long	
unsigned long	4字节
long long	
unsigned long long	8字节

- 十进制：[±]0～9
- 八进制：0开头
- 十六进制：0x(或0X)开头
- 后缀 L（或 l）为 长整型
- 后缀 U（或 u）为 无符号型

数据类型
——整型数据及取值范围

类型	说明符	位数	数值范围
短整	short	16	-32768～32767
基本	int	32	-2^{31}～$(2^{31}-1)$
长整	long	32	-2^{31}～$(2^{31}-1)$
长整	long long	64	-2^{63}～$(2^{63}-1)$
无符号			
unsigned	short	16	0～65535
unsigned	[int]	32	0～$(2^{32}-1)$
unsigned	long	32	0～$(2^{32}-1)$
unsigned	long long	64	0～$(2^{64}-1)$

数据类型

——实型数据

基本数据类型和表达式

```
#include <iostream>
using namespace std;
int main() {
   const int PRICE =
   int num, total;
   float v, r, h;
   num = 10;
   total = num * PRIC
   cout << total << e
   r = 2.5;
   h = 3.2;
   v = 3.14159 * r * r *
   cout << v << endl;
   return 0;
}
```

实型变量

float 4字节
$3.4\times10^{\pm38}$
7位有效数字
double 8字节
$1.7\times10^{\pm308}$
16位有效数字
long double
12字节

- 默认为double型
- 后缀 F（或 f）为 float型
- 后缀 L（或 l）为 long double型

实型常量

- 一般形式:12.5,-12.5
- 指数形式:2.345E-3
表示2.345×10^{-3}

数据类型
——字符型数据（一）

基本数据类型和表达式

- 字符常量
  - 单引号括起来的一个字符，
    如：‘a’, ‘D’, ‘?’, ‘$’
  - 转义字符： ‘\n’, ‘\t’, ‘\\’, ‘\’’, ‘\”’
- 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

数据类型
——字符型数据（二）

- 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。
- 字符串常量

例:"CHINA"　C H I N A \0

"a"　a \0

'a'　a

所以：char　c;

c="a";

数据类型
——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

数据类型
——变量的声明和定义

基本数据类型和表达式

- 变量在使用之前需要声明去类型和名称
- 变量声明语句的形式如下：

数据类型 变量名1, 变量名2, …, 变量名n;

- 变量的声明只是将变量名标识符的信息告诉编译器，并不一定引起内存的分配
- 变量的定义意味着给变量分配内存空间，变量名就是对应内存单元的名字
- C++程序中，大多数情况下变量声明就是定义，只有声明外部变量时例外

数据类型
——变量初始化

例：

```
int a = 3;
double f = 3.56;
char c = 'a';
int c(5);
```

数据类型
——混合运算时的类型转换

- 不同类型数据进行混合运算时，C++编译器会自动进行类型转换。
- 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。
- 可以采用强制类型转换：

例如：

```
float c;
int a, b;
c = float(a) / float(b);
或 c = (float)a / (float)b;
```

变量的存储类型

- auto
  - 属于一时性存储，其存储空间可以被若干变量多次覆盖使用。
- register
  - 存放在通用寄存器中。
- extern
  - 在所有函数和程序段中都可引用。
- static
  - 在内存中是以固定地址存放的，在整个程序运行期间都有效。

符号常量

const 数据类型 常量名=常量值;
或
数据类型 const 常量名=常量值;

```
const float PI = 3.1415926f;

const float PI;  //错！必须初始化
PI = 3.1415926f; //错！常量不能被赋值
```

运算符和表达式

- 表达式是计算求值的基本单位
  - 用于计算的公式，由运算符(例如：+, -, *, /)、运算数(也称操作数)和括号组成
- 表达式可以定义为：
  - 一个常量或标识对象的标识符是一个最简单的表达式，其值是常量或对象的值
  - 一个表达式的值可以用来参与其他操作，即用作其他运算符的操作数，这就形成了更复杂的表达式
  - 包含在括号中的表达式仍是一个表达式，其类型和值与未加括号时的表达式相同

运算符

- 有些运算符需要两个运算数，形式为：
  运算数1 运算符 运算数2
  这样的运算符称为二元运算符（或二目运算符）
- 只需要一个运算数的称为一元运算符（或单目运算符）
- C++语言中定义了丰富的运算符
  - 算术运算符、关系运算符、逻辑运算符
- 运算符具有优先级与结合性
  - 结合性：自左向右，自右向左

算术运算符与算术表达式

- 基本算术运算符

  + - * /(若整数相除，结果取整)

  %（取余，操作数为整数）

- 优先级与结合性

  先乘除，后加减，同级自左至右

- ++，--（自增、自减）

  例：i++; --j;

```
cout << i++;    // 输出i自增前的值
cout << ++i;    // 输出i自增后的值
```

赋值运算符和赋值表达式

简单的赋值运算符"="

- 举例

  n = n + 5
- 表达式的类型

  等号左边对象的类型
- 表达式的值

  等号左边对象被赋值后的值
- 结合性：自右向左

赋值运算符和赋值表达式

复合的赋值运算符

- 有10种复合运算符：

  +=, -=, *=, /=, %=,

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

- 优先级与“=”相同，结合性也是自右向左
- 例

  a += 3 等价于 a = a + 3

  x *= y + 8 等价于 x = x * (y + 8)

赋值运算符和赋值表达式
——赋值表达式举例

基本数据类型和表达式

a = 5　　表达式值为5

a = b = c = 5　　表达式值为5，a, b, c均为5

a = 5 + (c = 6) 表达式值为11，a为11，c为6

a = (b = 4) + (c = 6)

表达式值为10，a为10，b为4，c为6

a = (b = 10) / (c = 2)

表达式值为5，a为5，b为10，c为2

a += a -= a * a 相当于 a = a + (a = a - a * a)

逗号运算和逗号表达式

- 逗号也是一个运算符，格式为

  表达式1，表达式2

- 求解顺序及结果

  先求解1，再求解2，最终结果为表达式2的值

- 最低优先级，结合性自左向右
- 例

  a = 3 * 5, a * 4　　最终结果为60

关系运算与关系表达式

- 关系运算是比较简单的一种逻辑运算，优先次序为：

$$\underbrace{< \quad <= \quad > \quad >=}_{\text{优先级相同（高）}} \qquad \underbrace{== \quad !=}_{\text{优先级相同（低）}}$$

- 关系表达式是一种最简单的逻辑表达式

  其结果类型为 bool，值只能为 true 或 false。

- 例如：a > b，c <= a + b，x + y == 3

逻辑运算与逻辑表达式

- 逻辑运算符

  !(非)　&&(与)　||(或)

  优先次序：　高　→　低

- 逻辑表达式

  例如：(a > b) && (x > y)

  其结果类型为 bool，值只能为 true 或 false

逻辑运算与逻辑表达式

- "&&"的"短路特性"

  表达式1 && 表达式2

  - 先求解表达式1
  - 若表达式1的值为false，则最终结果为false，**不再求解表达式2**
  - 若表达式1的结果为true，则求解表达式2，以表达式2的结果作为最终结果
- "||"也具有类似的特性
  - (a==b)||(++c==1)

条件运算符与条件表达式

- 一般形式

  表达式1 ？ 表达式2 ： 表达式3

  表达式1 必须是 bool 类型

- 执行顺序
  - 先求解表达式1，
  - 若表达式1的值为true，则求解表达式2，表达式2的值为最终结果
  - 若表达式1的值为false，则求解表达式3，表达式3的值为最终结果
- 例： x = a > b ? a : b;

条件运算符与条件表达式

- 注意：
  - 条件运算符优先级高于赋值运算符，低于逻辑运算符
  - 表达式2、3的类型可以不同，条件表达式的最终类型为 2 和 3 中较高的类型。
- 例：

$$\underbrace{x = \underbrace{a > b ? a : b}_{①}}_{②};$$

sizeof 运算符

- 语法形式

  sizeof (类型名)

  或 sizeof 表达式
- 结果值：

  “类型名”所指定的类型或“表达式”的结果类型所占的字节数。
- 例：

```
sizeof(short)
sizeof x
```

位运算——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：
  - 将某一位置0，其他位不变。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——按位或（|）

- 运算规则
  - 将两个运算量的每一个位进行逻辑或操作
- 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 1 1 1
```

- 用途：
  - 将某些位置1，其他位不变。
    例如：将 int 型变量 a 的低字节置 1：

```
a = a | 0xff;
```

位运算——按位异或（^）

- 运算规则
  - 两个操作数进行异或：
    若对应位相同，则结果该位为 0，
    若对应位不同，则结果该位为 1，
- 举例：计算 071^052

071:		0 0 1 1 1 0 0 1
052:	(^)	0 0 1 0 1 0 1 0
071^052 :		0 0 0 1 0 0 1 1

位运算——按位异或（^）

- 用途：
  - 使特定位翻转（与0异或保持原值，与1异或取反）

    例如：要使 01111010 低四位翻转：

```
      0 1 1 1 1 0 1 0
(^)   0 0 0 0 1 1 1 1
      ---------------
      0 1 1 1 0 1 0 1
```

位运算——取反(~)

单目运算符，对一个二进制数按位取反。

例：　025：0000000000010101

~025：1111111111101010

位运算——移位

- 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”

```
-8 >> 2 = -2
  -8 : 11111000
  -2 : 11111110
```

运算符优先级

括号
++, --, sizeof
*, /, %
+, -
==, !=
位运算
&&
||
?:
赋值运算
逗号运算

高

低

混合运算时数据类型的转换

——隐含转换

- 一些二元运算符（算术运算符、关系运算符、逻辑运算符、位运算符和赋值运算符）要求两个操作数的类型一致。
- 在算术运算和关系运算中如果参与运算的操作数类型不一致，编译系统会自动对数据进行转换（即隐含转换），基本原则是将低类型数据转换为高类型数据。

char, short, int, unsigned, long, unsigned long, float, double

低 ——————————————————→ 高

混合运算时数据类型的转换

——隐含转换

- 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。

混合运算时数据类型的转换

——显式转换

- 语法形式（3种）：
  - 类型说明符(表达式)　　// C++风格
  - (类型说明符)表达式　　// C风格
  - 类型转换操作符<类型说明符>(表达式)
    - 类型转换操作符可以是：const_cast、dynamic_cast、reinterpret_cast、static_cast
- 显式类型转换的作用是将表达式的结果类型转换为类型说明符所指定的类型。
- 例：int(z)，(int)z，static_cast<int>(z)三种完全等价

语句

语句

- 程序的执行流程是由语句来控制的。C++的语句包括：
  - 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

表达式语句

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他表达式中，而语句不可。

  例如：if ((a = b) > 0) t = a;

  不可写为：if ((a = b;) > 0) t = a;

复合语句

语句

- 将多个语句用一对大括号包围，便构成一个复合语句（或语句块）
- 例如

```
{
    sum=sum+i;
    i++;
}
```

简单的输入、输出

- 向标准输出设备（显示器）输出

  例：
  ```
  int x;
  cout << "x=" << x;
  ```

- 从标准输入设备（键盘）输入

  例：
  ```
  int x;
  cin >> x;
  ```

算法的基本控制结构

算法的基本控制结构

- 顺序结构
- 分支结构
- 循环结构

如何解决分支问题？

例2-2

输入一个年份，判断是否闰年。

```
#include <iostream>
using namespace std;
int main() {
  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;
  return 0;
}
```

运行结果：

Enter the year: *2000*
2000 is a leap year

if 语句
——三种形式

算法的基本控制结构

```
if (表达式) 语句
    例: if (x > y) cout << x;
if (表达式) 语句1 else 语句2
    例: if (x > y) cout << x;
        else cout << y;
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
```

if 语句

算法的基本控制结构

if(x = 3)
和
if(x == 3)
的区别

3 == x

浮点数的相等判断

fabs(x-y) < fEpsilon

如何解决多分问题？

例2-3

输入两个整数，比较两个数的大小。

```
#include<iostream>
using namespace std;
int main() {
  int x, y;
  cout << "Enter x and y:";
  cin >> x >> y;
  if (x != y)
    if (x > y)
      cout << "x > y" << endl;
    else
      cout << "x < y" << endl;
  else
    cout << "x = y" << endl;
  return 0;
}
```

运行结果1：

```
Enter x and y:5 8
x < y
```

运行结果2：

```
Enter x and y:8 8
x = y
```

运行结果3：

```
Enter x and y:12 8
x > y
```

if 语句
——嵌套

算法的基本控制结构

- 一般形式

```
if(   )
    if(   ) 语句 1
      else 语句 2
else
    if(   ) 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。

特殊的多分支结构

例2-4

输入一个0～6的整数，转换成星期输出。

```
#include <iostream>
using namespace std;
int main() {
   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;
   }
   return 0;
}
```

switch 语句

算法的基本控制结构

- 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
  }
```

可以是整型、字符型、枚举型

每个常量表达式的值不能相同，次序不影响执行结果。

可以是多个语句，但不必用{ }。

- 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。因此，每个case分支最后应该加break语句。

使用switch语句应注意的问题

- case分支可包含多个语句，且不用{ }。
- 表达式、判断值都是int型或char型。
- 若干分支执行内容相同可共用一组语句。

如何有效地完成重复工作

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;
int main() {
  int i = 1, sum = 0;
  while (i <= 10) {
      sum += i;  //相当于sum = sum + i;
      i++;
  }
  cout << "sum = " << sum << endl;
  return 0;
}
```

运行结果：

sum = 55

while 语句

算法的基本控制结构

- 形式

  while （表达式） 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。

- 执行顺序

  先判断表达式的值，若为 true 时，执行语句。

先执行循环体，后判断条件的情况

例2-6

输入一个整数，将各位数字反转后输出。

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

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

运行结果：

```
Enter the number: 365
The number in reverse order is  563
```

算法的基本控制结构

do-while 语句

- 一般形式

  do　　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。

  while（表达式）

- 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

对比下列程序：i > 10?

算法的基本控制结构

程序1：

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

程序2：

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

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

为true时执行循环体

每次执行完循环体后求解

初始语句可以是表达式语句或声明语句

表达式2一般用于改变循环控制变量的值

for 语句

- 初始语句、表达式1和表达式2都可以省略，分号不能省略

```
for (;;) 语句      //相当于while(true)语句
```

- 表达式1是循环控制条件
- 当初始语句是一个声明语句时，可定义循环变量并为它赋初值。在初始语句中声明的变量，只在循环内部有效

```
for (int i = 1; i <= 100; i++)
    sum = sum + i;
```

for 语句

- 当初始语句是表达式语句时，可以是简单表达式，也可以是逗号表达式

```
for (i = 0, j = 100; i <= j; i++, j--)
    k = i + j;
```

- 表达式2一般用于改变循环控制变量的值
- 如果省略表达式1和3，只有表达式2，则完全等同于while语句

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

例2-8

输入一个整数，求出它的所有因子。

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

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

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

运行结果1：

```
Enter a positive integer: 36
Number  36   Factors  1  2  3  4  6  9  12  18  36
```

运行结果2：

```
Enter a positive integer: 7
Number  7   Factors  1  7
```

例2-9 编写程序输出以下图案

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

```
#include <iostream>
using namespace std;

int main() {
  const int N = 4;
  for (int i = 1; i <= N; i++) { //输出前4行图案
    for (int j = 1; j <= 30; j++)
      cout << ' ';    //在图案左侧空30列
    for (int j = 1; j <= 8 - 2 * i; j++)
      cout << ' ';
    for (int j = 1; j <= 2 * i - 1; j++)
      cout << '*';
    cout << endl;
  }
```

```
  //输出后3行图案
  for (int i = 1; i <= N - 1; i++) {
    for (int j = 1; j <= 30; j++)
      cout << ' ';   //在图案左侧空30列
    for (int j = 1; j <= 7 - 2 * i ;j++)
      cout << '*';
    cout << endl;
  }
  return 0;
}
```

循环结构与选择结构相互嵌套

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

例2-10

- 读入一系列整数，统计出正整数个数i和负整数个数j,读入0则结束。
- 分析：
  - 需要读入一系列整数，但是整数个数不定，要在每次读入之后进行判断，因此使用while循环最为合适。循环控制条件应该是n!=0。由于要判断数的正负并分别进行统计，所以需要在循环内部嵌入选择结构。

```
#include <iostream>
using namespace std;

int main() {
   int i = 0, j = 0, n;
   cout << "Enter some integers please (enter 0 to quit): "
        << endl;
   cin >> n;
   while (n != 0) {
     if (n > 0) i += 1;
     if (n < 0) j += 1;
     cin >> n;
   }
   cout << "Count of positive integers: " << i << endl;
   cout << "Count of negative integers: " << j << endl;
   return 0;
}
```

break 和 continue 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。

typedef语句

自定义数据类型

- 为一个已有的数据类型另外命名
- 语法形式

  typedef　已有类型名　新类型名表;
- 例如

```
typedef double Area, Volume;
typedef int Natural;
Natural i1, i2;
Area a;
Volume v;
```

枚举类型—enum

自定义数据类型

- 只要将需要的变量值一一列举出来，便构成了一个枚举类型。
- 枚举类型的声明形式如下：

  enum　枚举类型名　{变量值列表};

- 例如：

```
enum Weekday
{SUN, MON, TUE, WED, THU, FRI, SAT};
```

枚举类型—enum

自定义数据类型

- 枚举类型应用说明：
  - 对枚举元素按常量处理，不能对它们赋值。例如，不能写：SUN = 0;
  - 枚举元素具有默认值，它们依次为：0,1,2,.....。
  - 也可以在声明时另行指定枚举元素的值，如：

    enum Weekday{SUN=7,MON=1,TUE,WED,THU,FRI,SAT};
  - 枚举值可以进行关系运算。
  - 整数值不能直接赋给枚举变量，如需要将整数赋值给枚举变量，应进行强制类型转换。

例2-11

自定义数据类型

- 设某次体育比赛的结果有四种可能：胜（WIN）、负（LOSE）、平局（TIE）、比赛取消（CANCEL），编写程序顺序输出这四种情况。
- 分析：由于比赛结果只有四种可能，所以可以声明一个枚举类型，声明一个枚举类型的变量来存放比赛结果。

```
#include <iostream>
using namespace std;
enum GameResult {WIN, LOSE, TIE, CANCEL};
int main() {
   GameResult result;
   enum GameResult omit = CANCEL;
   for (int count = WIN; count <= CANCEL; count++) {
     result = GameResult(count);
     if (result == omit)
       cout << "The game was cancelled" << endl;
     else {
       cout << "The game was played ";
       if (result == WIN)
         cout << "and we won!";
       if (result == LOSE)
         cout << "and we lost.";
       cout << endl;
     }
   }
   return 0;
}
```

运行结果

```
The game was played and we won!
The game was played and we lost.
The game was played
The game was cancelled
```

结构体——结构的声明

自定义数据类型

- 结构的概念

  结构是由不同数据类型的数据组成的集合体。

- 声明结构类型

```
struct   结构名
{
        数据类型    成员名 1;
        数据类型    成员名 2;
          ┆
        数据类型    成员名 n;
};
```

结构体——结构的声明

自定义数据类型

- 举例：

```
struct Student {  //学生信息结构体
    int num;        //学号
    string name;    //姓名
    char gender;    //性别
    int age;        //年龄
    float score;    //成绩
    string addr;    //住址
};
```

结构体——结构变量说明

自定义数据类型

- 变量说明形式

  结构名　结构变量名;

- 注意:
  - 结构变量的存储类型概念、它的寿命、可见性及使用范围与普通变量完全一致。
  - 结构变量说明在结构类型声明之后，二者也可同时进行。
  - 结构变量占内存大小可用 sizeof 运算求出:

    sizeof(运算量)

结构体
——结构变量的初始化和使用

自定义数据类型

- 初始化

  说明结构变量的同时可以直接设置初值。

- 使用

  结构体成员的引用形式：

  结构变量名.成员名

例2-12

自定义数据类型

结构体变量的初始化和使用

```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;
struct Student { //学生信息结构体
  int num;//学号
  string name;//姓名
  char gender;//性别
  int age;//年龄
} stu = { 97001, "Lin Lin", 'F', 19 };
int main() {
  cout << setw(7) << stu.num << setw(20) << stu.name
       << setw(3) << stu.sex << setw(3)  << stu.age;
  return 0;
}
```

运行结果:

```
  97001        Lin Lin   F 19
```

变量是如何工作的？

深度探索

- 站在目标代码的角度上看问题
- 变量具有两个重要属性——数据类型和变量名
  - 它们都用文字表示，便于人们理解，但不便于CPU识别，因此它们不能出现在目标代码中；
  - 下面将讨论它们在目标代码中的表示方式。

变量是如何工作的？

深度探索

- 变量名
  - 源代码中，变量依靠变量名来标识；
  - 目标代码中，变量依靠地址来标识，每个变量的地址互不相同。

```
int a, b;
int main() {
  a++;
  b++;
  return 0;
}
```

```
incl 0x80495f8
incl 0x80495fc
```

地址

变量是如何工作的？

深度探索

- 数据类型
  - 任何数据在内存中都是用二进制串的形式表示的；
  - 一串二进制数，只有确定了类型，才有真实的含义；
  - 例：10111111,10000000,00000000,00000000
    - 解释为int型：-1,082,130,432
    - 解释为unsigned型：3,212,836,864
    - 解释为float型：-1.0

变量是如何工作的？

深度探索

int a=-1082130432; //10111111 10000000 00000000 00000000

unsigned b=3212836864;//10111111 10000000 00000000 00000000

a++; // 10111111 10000000 00000000 00000001，即-1 082 130 431

b++; // 10111111 10000000 00000000 00000001，即3 212 836 865

a和b类型不同，但相同操作后得到相同的二进制结果

变量是如何工作的？

深度探索

- 目标代码体现数据类型的方式
  - 类型的特性蕴含于操作之中；
  - 对不同数据类型，源代码中形式上相同的操作，会被转化为目标代码中的不同操作。

```
int a;
short b;
char c;
int main() {
  a++;
  b++;
  c++;
  return 0;
```

```
incl 0x80495f8
incw 0x80495fc
incb 0x80495fe
```

不同的操作

变量是如何工作的？

深度探索

- 变量的声明
  - 将一个标识符声明为变量，告诉编译器这个标识符表示变量，同时还指出了它的数据类型；
  - 只有确定了一个变量的数据类型，变量参与的操作才具有完整的意义，编译器才能将该变量参与的表达式翻译为合适的操作。
- 变量的定义
  - 不仅确定了一个标识符表示变量，以及该变量的类型，还确定了变量地址的分配位置。

表达式的执行原理

深度探索

- 什么是寄存器？
  - CPU内部的存储单元
  - 读写速度非常快
  - 数量很少
    - IA-32有eax、ebx、ecx、edx、esp、ebp、esi、edi八个通用寄存器
- 大部分CPU指令都需要读写寄存器。
- 复杂的表达式，分步执行，每条指令只能做一次基本运算，中间结果暂存在寄存器中。

表达式的执行原理

深度探索

```
int a, b, c, d;
int e;
int main() {
  a = 4;
  b = 2;
  c = 1;
  d = 10;
  e = (a + b) * (c - d);
  return 0;
}
```

把4存入地址0x80495d8中

```
movl    $0x4,0x80495d8
movl    $0x2,0x80495dc
movl    $0x1,0x80495e0
movl    $0xa,0x80495e4
```

0x80495d8：a变量
0x80495dc：b变量
0x80495e0：c变量
0x80495e4：d变量
0x80495e8：e变量

表达式的执行原理

深度探索

```
int a, b, c, d;
int e;
int main()
{
    a = 4;
    b = 2;
    c = 1;
    d = 10;
    e = (a + b) * (c - d);
    return 0;
}
```

把0x80495dc地址的值放到eax寄存器中

ecx←ecx + eax

eax←eax - eax

```
mov     0x80495dc,%eax
mov     0x80495d8,%ecx
add     %eax,%ecx
mov     0x80495e4,%edx
mov     0x80495e0,%eax
sub     %edx,%eax
imul    %ecx,%eax
mov     %eax,0x80495e8
```

eax←(eax * eax)的低32位

0x80495d8：a变量
0x80495dc：b变量
0x80495e0：c变量
0x80495e4：d变量
0x80495e8：e变量

小结与复习建议

- 主要内容
  - 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、自定义数据类型
- 达到的目标
  - 掌握C++语言的基本概念和基本语句，能够编写简单的程序段。