第 2 章

本章主要内容

计算机的最基本功能是数据处理

- C++支持的基本数据类型：
  - 整数、实数、字符、布尔数据
- C++支持的基本运算
  - 算术运算、关系运算、逻辑运算、位运算、逗号运算、条件运算

程序要能够输入数据、输出数据

- C++中的数据输入/输出可以调用预定义的功能模块实现

程序的执行流程

- 顺序的，因此程序要能够
- 对执行流程进行选择（选择、开关语句）；
- 反复用同一算法依次处理大批量数据（循环语句）。

枚举类型

- 通过列出所有可取值来定义一种新类型。

学习要求：

- 学习视频课件；
- 阅读教材第 2 章；
- 按照实验指导视频或者《C++语言程序设计（第 4 版）学生用书》完成实验；
- 完成在线选择题和编程题；

参考教材

- 《C++语言程序设计》（第 4 版），郑莉等编著，清华大学出版社出版
- 《C++语言程序设计（第 4 版）学生用书》，郑莉等编著，清华大学出版社出版

C++的特点和程序实例

C++的产生和发展

- 从 C 语言发展演变而来，最初被称为"带类的 C"；

- 1983 年正式取名为 C++；
- 1998 年 11 月被国际标准化组织（ISO）批准为国际标准；
- 2003 年 10 月 15 日发布第 2 版 C++标准 ISO/IEC 14882:2003；
- 2011 年 8 月 12 日 ISO 公布了第三版 C++标准 C++11，包含核心语言的新机能、扩展 C++标准程序库。
- 2014 年 8 月 18 日 ISO 公布了 C++14，其正式名称为"International Standard ISO/IEC 14882:2014(E) Programming Language C++"。
- C++14 作为 C++11 的一个小扩展，主要提供漏洞修复和小的改进。

C++的特点

- 兼容 C，支持面向过程的程序设计；
- 支持面向对象的方法；
- 支持泛型程序设计方法。

命名空间

避免命名冲突

std 是 C++标准库的命名空间（ namespace）名

using namespace std 表示打开 std 命名空间

例 2-1

```
#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++字符集和词法记号

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

- 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
- 分隔符 () {} , : ;
  用于分隔各个词法记号或程序正文
- 运算符（操作符） 用于实现各种运算的符号
- 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

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++关键字或操作符。

基本数据类型、常量、变量

C++能够处理的基本数据类型

- 整数类型；
- 浮点数类型；
- 字符类型；
- 布尔类型。

程序中的数据

- 常量
  - 在源程序中直接写明的数据；
  - 其值在整个程序运行期间不可改变。
- 变量
  - 在程序运行过程中允许改变的数据。

整数类型

- 基本的整数类型：int
- 按符号分
  - 符号的（signed）
  - 无符号的（unsigned）
- 按照数据范围分
  - 短整数（short）
  - 长整数（long）
  - 长长整数（ long long ）
- ISO C++标准并没有明确规定每种数据类型的字节数和取值范围，它只是规定它们之间的字节数大小顺序满足：
  (signed/unsigned)signed char ≤ (unsigned) short int ≤ (unsigned) int ≤ (unsigned) long int ≤ long long int

字符类型 （char）

- 容纳单个字符的编码；
- 实质上存储的也是整数。

浮点数类型

- 单精度（float）
- 双精度（double）
- 扩展精度（long double）

字符串类型（详见第 6 章）

- 有字符串常量
- 基本类型中没有字符串变量

- 采用字符数组存储字符串（C 风格的字符串）
- 标准 C++类库中的 String 类（C++风格的字符串）

布尔类型（bool）

- 只有两个值：true（真）、false（假）
- 常用来表示关系比较、相等比较或逻辑运算的结果

各基本类型的取值范围

类型名	长度（字节）	取值范围
bool	1	false，true
char	1	-128~127
signed char	1	-128~127
unsigned char	1	0~255
short（signed short）	2	-32768~32767
unsigned short	2	0~65535
int（signed int）	4	-2^{31}~2^{31}-1
unsigned int	4	0~2^{32}-1
long（signed long）	4	-2^{31}~2^{31}-1
unsigned long	4	0~2^{32}-1
long long	8	-2^{63}~2^{63}-1
unsigned long long	8	0~2^{64}-1
float	4	绝对值范围3.4×10^{-38}~3.4×10^{38}
double	8	绝对值范围1.7×10^{-308}~1.7×10^{308}
long double	8	绝对值范围1.7×10^{-308}~1.7×10^{308}

注：表中各类型的长度和取值范围，以面向IA32处理器的msvc12和2 gcc4.8为准

常量

- 在程序运行的整个过程中其值始终不可改变的量；
- 直接使用符号（文字）表示的值；
- 例如：12，3.5，'A'都是常量。

整数常量

- 以文字形式出现的整数；
- 十进制
  - 若干个 0~9 的数字，但数字部分不能以 0 开头，正数前边的正号可以省略。
- 八进制
  - 前导 0+若干个 0~7 的数字。
- 十六进制
  - 前导 0x+若干个 0~9 的数字及 A~F 的字母（大小写均可）。
- 后缀

- 后缀 L（或 l）表示类型至少是 long，后缀 LL（或 ll）表示类型是 long long，后缀 U（或 u）表示 unsigned 类型。

浮点数常量

- 以文字形式出现的实数；
- 一般形式：
  - 例如，12.5，-12.5 等。
- 指数形式：
  - 例如，0.345E+2，-34.4E-3；
  - 整数部分和小数部分可以省略其一。
- 浮点常量默认为 double 型，如果后缀 F（或 f）可以使其成为 float 型，例如：12.3f。

字符常量

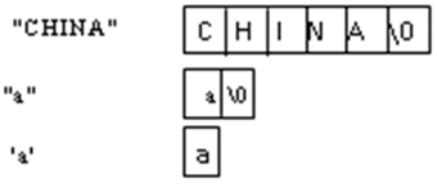
- 单引号括起来的一个字符，如：'a'、'D'、'?'、'$'；
- C++转义字符列表（用于在程序中表示不可显示字符）

字符常量形式	ASCII码（十六进制）	含义
\a	07	响铃
\n	0A	换行
\t	09	水平制表符
\v	0B	垂直制表符
\b	08	退格
\r	0D	回车
\f	0C	换页
\\	5C	字符“\”
\”	22	双引号
\’	27	单引号
\?	3F	问号

C 风格字符串常量

- 一对双引号括起来的字符序列；
- 在内存中按串中字符的排列次序顺序存放，每个字符占一个字节；
- 在末尾添加 ‘\0’ 作为结尾标记。

例：



通过添加前缀可以改变字符常量或者字符串常量的类型，前缀及其含义如下表所示：

前缀	含义	类型
u	Unicode 16 字符	char16_t
U	Unicode 32字符	char32_t
L	宽字符	wchar_t
u8	UTF-8（仅用于字符串字面常量）	char

变量：在程序的运行过程中，其值可变的量

- 变量定义
  - 数据类型　变量名 1, 变量名 2, ..., 变量名 n;
- 初始化
  - C++语言中提供了多种初始化方式；
  - 例如：

    int a = 0;

    int a(0);

    int a = {0};

    int a{0};

    其中使用大括号的初始化方式称为列表初始化，列表初始化时不允许信息的丢失。

    例如用 double 值初始化 int 变量，就会造成数据丢失。

符号常量

- 常量定义语句的形式为：
  - const 数据类型说明符 常量名=常量值；

  或：
  - 数据类型说明符 const 常量名=常量值；
- 例如，可以定义一个代表圆周率的符号常量：
  - const float PI = 3.1415926；
- 符号常量在定义时一定要初始化，在程序中间不能改变其值。

程序举例

题目：读入并显示整数

- 主要知识点：
  - 常量
    - 在源程序中直接写明的数据，其值在整个程序运行期间不可改变，这样的数据称为常量。
  - 变量
    - 在运行过程中从计算机的外部设备（例如键盘、硬盘）读取的，这些数据的值在程序运行过程中允许改变，这样的数据称为变量
  - 从键盘输入数据
    - iostream 类的对象 cin 的>>操作，可以从标准输入设备（通常是键盘）读入数据
  - 数据的存储
    - 为了存储数据，需要预先为这些数据分配内存空间。
    - 变量的定义就是在给变量命名的时候分配内存空间。
- 源代码

```
#include <iostream>
using namespace std;
int main()
{
    int radius;
    cout<<"Please enter the radius!\n";
    cin>>radius;
    cout<< “The radius is:” <<radius<< ‘\n’ ;
        cout<< “PI is:” <<3.14<< ‘\n’ ;
    cout<<"Please enter a different radius!\n";
    cin>>radius;
    cout<<"Now the radius is changed to:"
         <<radius<< ‘\n’ ;
    return 0;
}
```

题目：为常量命名

- 主要知识点：符号常量
- 源代码

```
#include <iostream>
using namespace std;
int main()
{   const double pi(3.14159);
    int radius;
    cout<<"Please enter the radius!\n";
    cin>>radius;
    cout<<"The radius is:"<<radius<<'\n';
    cout<<"PI is:"<<pi<<'\n';
    cout<<"Please enter a different radius!\n";
    cin>>radius;
    cout<<"Now the radius is changed to:"<<radius<<'\n';
    cout<<"PI is still:"<<pi<<'\n  ';
    //cin>>pi;
    return 0;
}
```

- 运行结果：

```
Please enter the radius!
2
The radius is:2
PI is:3.14159
Please enter a different radius!
3
Now the radius is changed to:3
PI is still:3.14159
```

题目：变量的初始化

- 主要知识点：变量的初始化
  - 虽然变量的值是可以在运行时获得的，但是在定义变量时也可以进行初始化，而且应该提倡进行初始化；

- 未经初始化的变量，其值可能是随机的。如果误用了未经初始化也没有给予确定值的变量，就会引起错误。

- 源代码

```
#include <iostream>
using namespace std;
int main()
{
    const double pi(3.14159);
    int radius(0);
    cout<<"The initial radius is:"<<radius<<'\n';
    cout<<"PI is:"<<pi<<'\n  '; 。
    cout<<"Please enter a different radius!\n";
    cin>>radius;
    cout<<"Now the radius is changed to:"<<radius<<'\n';
    cout<<  "PI is still:"  <<pi<<  '\n'  ;
    return 0;
}
```

算术运算与赋值运算

算术运算

- 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;

赋值运算

- 将值赋给变量
- 赋值运算符“=”

- 赋值表达式：
  - 用赋值运算符连接的表达式
  - 例：

    n=5

    n = n + 5
  - 表达式的值

    赋值运算符左边对象被赋值后的值
  - 表达式的类型

    赋值运算符左边对象的类型
- 复合的赋值运算符
  - +=，-=，*=，/=，%=，<<=，>>=，&=，^=，|=
  - 例

    a += 3 等价于 a = a + 3

    x *= y + 8 等价于 x = x * (y + 8)

逗号运算、关系运算、逻辑运算和条件运算

逗号运算和逗号表达式

- 格式

  表达式 1，表达式 2
- 求解顺序及结果
  - 先求解表达式 1，再求解表达式 2
  - 最终结果为表达式 2 的值
- 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：

  < <= > >=　　== !=

  优先级相同（高）　　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
- 逻辑运算结果类型：bool，值只能为 true 或 false
- 逻辑表达式

  例如：(a > b) && (x > y)

“&&”（逻辑与）运算

- “&&”的运算规则
  - 两侧表达式都为真，结果为真；
  - 有一侧表达式为假，结果为假。
- “&&” 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

“||”（逻辑或）运算

- “||”的运算规则
  - 两侧表达式都为假，结果为假；
  - 有一侧表达式为真，结果为真。
- “||” 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 true，则最终结果为 true，不再求解表达式 2
  - 若表达式 1 的结果为 false，则求解表达式 2，以表达式 2 的结果作为最终结果

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;
```

如果 a>b 为 false，计算

如果 a>b 为 true，计算

条件运算的优先级

- 条件运算符优先级高于赋值运算符，低于逻辑运算符
  - 例

    x = a > b ? a : b;

    ① ②
- 表达式 1 是 bool 类型，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。

Sizeof 运算、位运算

sizeof 运算

- 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:   00000011
  5:   00000101
3&5:   00000001
```

- 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;

    1111 1110
  - 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;

    1111 1111

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:   00000011
  5:   00000101
3|5:   00000111
```

- 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:   00111001
  052:   00101010
071^052: 00010011
```

- 用途举例：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：

```
    01111010
(^) 00001111
    01110101
```

位运算——取反（~）

- 运算规则
  - 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”

运算优先级、类型转换

运算符优先级

优先级	运算符	结合性
1	[ ] ( ) . -> 后置 ++ 后置 —	左→右
2	前置 ++ 前置 — sizeof & * +（正号） -（负号） ~ ！	右→左
3	（强制转换类型）	右→左
4	.* ->*	左→右
5	* / %	左→右
6	+ -	左→右
7	<< >>	左→右
8	< > <= >=	左→右
9	== !=	左→右
10	&	左→右
11	^	左→右
12	|	左→右
13	&&	左→右
14	||	左→右
15	? :	右→左
16	= *= /= %= += -= <<= >>=&= ^= |=	右→左
17	,	左→右

混合运算时数据类型的转换

- 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。

数据的输入和输出

I/O 流

- 在 C++中，将数据从一个对象到另一个对象的流动抽象为“流”。流在使用前要被建立，使用后要被删除。
- 数据的输入与输出是通过 I/O 流来实现的，cin 和 cout 是预定义的流类对象。cin 用来处理标准输入，即键盘输入。cout 用来处理标准输出，即屏幕输出。
- 从流中获取数据的操作称为提取操作，向流中添加数据的操作称为插入操作。

预定义的插入符和提取符

- “<<”是预定义的插入符，作用在流类对象 cout 上便可以实现项标准输出设备输出。
  - cout << 表达式 << 表达式...
- 标准输入是将提取符作用在流类对象 cin 上。
  - cin >> 表达式 >> 表达式...
- 提取符可以连续写多个，每个后面跟一个表达式，该表达式通常是用于存放输入值的变量。例如：
  - int a, b;
  - cin >> a >> b;

常用的 I/O 流类库操纵符

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
setsprecision(int)	设置浮点数的小数位数（包括小数点）
setw(int)	设置域宽

例：cout << setw(5) << setprecision(3) << 3.1415;

if 语句

If 语句的语法形式

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

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;
}
```

嵌套的 if 结构

- 语法形式

```
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。

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;
}
```

switch 语句

- 语法形式

```
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
}
```

- 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。
- 注意
  - 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。

- 如果若干分支执行内容相同可共用一组语句。

例 2-4：输入一个 0～6 的整数，转换成星期输出

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

while 语句

- 语法形式

  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。

例 2-5 求自然数 1～10 之和

```
#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;
}
```

do-while 语句

- do-while 语句的语法形式

  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。

例 2-6：输入一个数，将各位数字翻转后输出

```
#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;
}
```

例 2-7 用 do-while 语句编程，求自然数 1~10 之和

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

对比下面的程序

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

- 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

- for 语句的另一种形式：范围 for 语句：

for（声明：表达式)
　语句

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

嵌套的控制结构、其他控制语句

例 2-10 输入一系列整数，统计出正整数个数 i 和负整数个数 j,读入 0 则结束。

```
#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;
}
```

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
  使程序的执行流程跳转到语句标号所指定的语句。不提倡使用。

自定义类型

类型别名：为已有类型另外命名

- typedef 已有类型名 新类型名表
  - 例：

    typedef double Area, Volume;

    typedef int Natural;

    Natural i1,i2;

    Area a;

    Volume v;
- using 新类型名 = 已有类型名;
  - 例：

    using Area = double;

    using Volume = double;

枚举类型

- 定义方式：

  将全部可取值一一列举出来。
- 语法形式：

  enum 枚举类型名 {变量值列表};

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

  默认情况下

  SUN=0，MON=1，TUE=2，……，SAT=6

C++包含两种枚举类型：

- 不限定作用域枚举类型：

  enum 枚举类型名 {变量值列表};
- 限定作用域的 enum 类将在第 4 章介绍。

不限定作用域枚举类型说明：

- 枚举元素是常量，不能对它们赋值

  例如有如下定义

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

  不能写赋值表达式：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};
- 也可以在声明时另行指定枚举元素的值；
- 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。
- 枚举值可以赋给整型变量。

例 2-11

- 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：
  比赛结果只有四种可能，可以声明一个枚举类型。

```
#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;
}
```

auto 类型与 decltype 类型

- auto：编译器通过初始值自动推断变量的类型
  - 例如：auto val = val1 + val2;
    如果 val1+val2 是 int 类型，则 val 是 int 类型；
    如果 val1+val2 是 double 类型，则 val 是 double 类型。
- decltype：定义一个变量与某一表达式的类型相同，但并不用该表达式初始化变量
  - 例如：decltype(i) j = 2;

小结

- 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。