第2章变量和基本类型

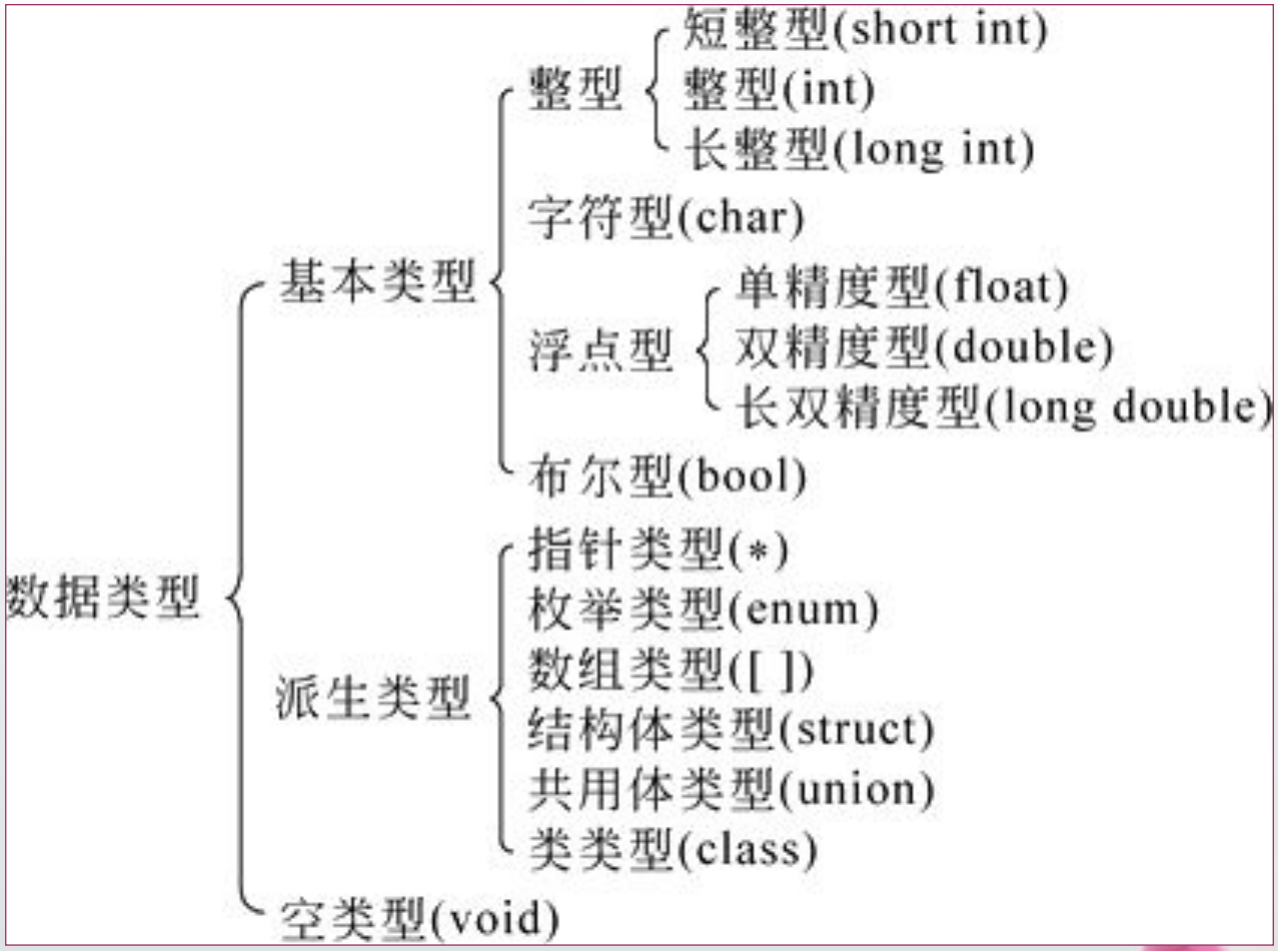
——C++程序设计

对外经济贸易大学 雷擎
leiqing@uibe.edu.cn

内容

- 2.1 基本内置类型
- 2.2 变量
- 2.3 复合类型
- 2.4 const限定符

C++可以使用的数据类型


数据类型
基本类型
整型
短整型(short int)
整型(int)
长整型(long int)
字符型(char)
浮点型
单精度型(float)
双精度型(double)
长双精度型(long double)
布尔型(bool)
派生类型
指针类型(*)
枚举类型(enum)
数组类型([ ])
结构体类型(struct)
共用体类型(union)
类类型(class)
空类型(void)

数值型和字符型数据的字节数

0 bit 位

000000000 1 byte = 8 bits

- 见code2.1

Type	Meaning	Minimum Size
bool	boolean	NA
char	character	8 bits
wchar_t	wide character	16 bits
char16_t	Unicode character	16 bits
char32_t	Unicode character	32 bits
short	short integer	16 bits
int	integer	16 bits
long	long integer	32 bits
long long	long integer	64 bits
float	single-precision floating-point	6 significant digits
double	double-precision floating-point	10 significant digits
long double	extended-precision floating-point	10 significant digits

000000000 表示整数的时候，最大值是 (2^7 -1)

011111111 = 10000000 -1 = (2^7) -1

11 = 2^1 + 2^0

2.1.3 字面值常量

- 数值常量
  - 整型:10, 010, 0x10;
  - 浮点数: 3.1415, 3.1415E0, 0.
- 字符和字符串常量
  - 'a'　单引号，只能是字符
  - "Hello C++"
- 符号常量
  - #define PRICE 30

2.1.2 类型转换

- 在表达式中不同类型的数据会自动地转换类型，以进行运算。还可以利用强制类型转换运算符将一个表达式转换成所需类型。

```
#include <iostream>
using namespace std;
int main( ){
  double x;
  int i;
  x=3.6;
  i=(int)x; //i=x;
  cout<<"x="<<x<<"， i="<< i<<endl;
  return 0;
}
```

2.2.1 变量定义

- 在C++语言中，要求对所有用到的变量作强制定义，也就是必须“先定义，后使用”。定义变量的一般形式是：

  变量类型 变量名表列;

- 变量名表列指的是一个或多个变量名的序列。如：

  float a,b,c,d,e;

为变量赋初值

- 允许在定义变量时对它赋予一个初值，这称为变量初始化。初值可以是常量，也可以是一个有确定值的表达式。

```
float a, b=5.78*3.5, c=2*sin(2.0);
```

- 若对变量未赋初值，则该变量的初值是一个不可预测的值。

```
int a, b;
cout<<a<<" "<<b<<" "<<c<<endl;//a,b值不定
```

2.2.2 变量的声明和定义

- 声明（extern标记）：使用其他程序定义的变量

  extern int i;

- 程序自身声明并定义变量

  int i;

- 给extern关键字标记的变量赋初值，就变成了定义

  extern int i = 10；//定义

- 在函数内部初始化一个有extern标记的变量会引发错误。

2.2.3 标识符

选择题

- 和其他高级语言一样，用来标识变量、符号常量、函数、数组、类型等实体名字的有效字符序列称为标识符(identifier)。简单地说，标识符就是一个名字。

- C++规定标识符只能由字母、数字和下划线3种字符组成，且第一个字符必须为字母或下划线。

- 注意：在C++中，大写字母和小写字母被认为是两个不同的字符；不能使用关键字作为标识符。

- C++没有规定标识符的长度(字符个数)，但各个具体的C编译系统都有自己的规定。有的系统取32个字符，超过的字符不被识别。

2.2.3 关键字

alignas	continue	friend	register	true
alignof	decltype	goto	reinterpret_cast	try
asm	default	if	return	typedef
auto	delete	inline	short	typeid
bool	do	int	signed	typename
break	double	long	sizeof	union
case	dynamic_cast	mutable	static	unsigned
catch	else	namespace	static_assert	using
char	enum	new	static_cast	virtual
char16_t	explicit	noexcept	struct	void
char32_t	export	nullptr	switch	volatile
class	extern	operator	template	wchar_t
const	false	private	this	while
constexpr	float	protected	thread_local	
const_cast	for	public	throw	

2.2.4 名字作用域(scope)

- 作用域
  - 名字有效的区域
- 全局作用域
  - 整个程序范围内有效
- 块作用域
  - 在其定义的{}程序块范围内有效
- 嵌套作用域
  - 内层作用域：被包含
  - 外层作用域：包含其他

2.2.4 名字作用域(scope)

全局作用域(global scope)

main的作用域(变量block scope)

for的作用域(变量block scope)

```
#include <iostream>
int main(){
    int sum = 0;
    // sum values from 1 through 10 inclusive

    for (int val = 1; val <= 10; ++val)
    sum += val; // equivalent to sum = sum + val

    std::cout << "Sum of 1 to 10 inclusive is "<< sum << std::endl;
    return 0;
}
```

2.2.4 名字作用域(scope)

外层作用域(outer scope)

内层作用域(inner scope)

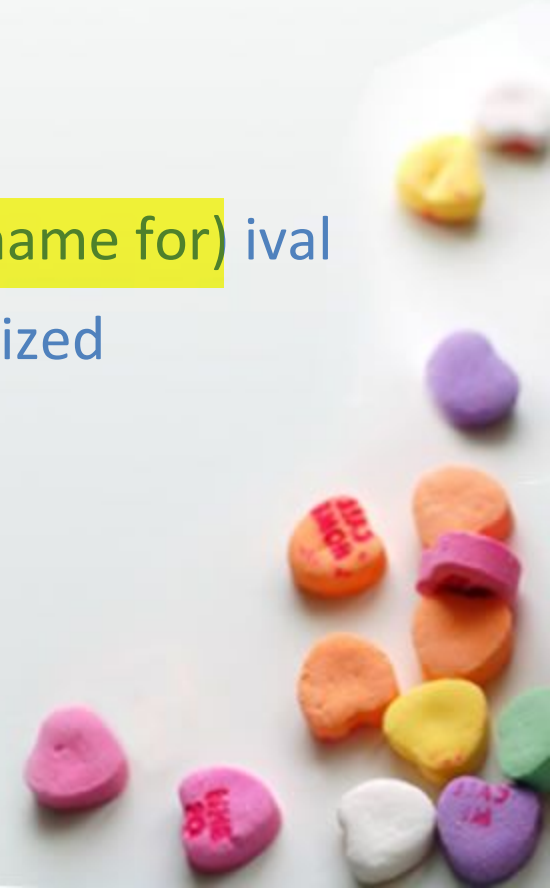
```
int reused = 42; // reused has global scope
int main(){
    int unique = 0; // unique has block scope
    // output #1: uses global reused; prints 42 0
    std::cout << reused << " " << unique << std::endl;
    int reused = 0; // new, local object named reused hides global reused
    // output #2: uses local reused; prints 0 0
    std::cout << reused << " " << unique << std::endl;
    // output #3: explicitly requests the global reused; prints 42 0
    std::cout << ::reused << " " << unique << std::endl;
    return 0;
}
```

2.3.1 引用(reference)

- 使用&d的形式定义

- 语法：

```
int ival = 1024;
int &refVal = ival; // refVal refers to (is another name for) ival
int &refVal2; // error: a reference must be initialized
```



引用的特点

- 引用是对象的别名，它在逻辑上不是独立的。

- 在定义引用时，程序把引用与它的初始值绑定（bind），而且其引用的对象在其整个生命周期中是不能被改变的（自始至终只能依附于同一个变量）。则任何对引用的操作即对变量的操作。

引用的使用

- 引用一般是用于处理函数的参数与返回值

- 引用的使用规则：
  - a：引用在创建的时候必须被初始化（指针可以在任何时候赋值）
  - b：必须与一个确定的合法内存单元相关联。不存在NULL引用
  - c：一旦引用初始化后，就不能改变引用所指向的变量。

&refval = val

绑定在一起的影子，扔也扔不掉！

不可申请同名引用：

```
int &refx = x;
int &refx = y;
```

错误！

```
int main() {
    int i = 10;   // A simple integer variable
    int &j = i;   // A Reference to the variable i
    j++;   // Incrementing j will increment both i and j.
    // check by printing values of i and j
    cout<< i << j <<endl; // should print 11 11
    // Now try to print the address of both variables i and j
    cout<< &i << &j <<endl;
    // surprisingly both print the same address and make us feel
that they are  alias to the same memory location.
    // In example below we will see what is the reality
    return 0;
}
```

2.3.2 指针(pointer)

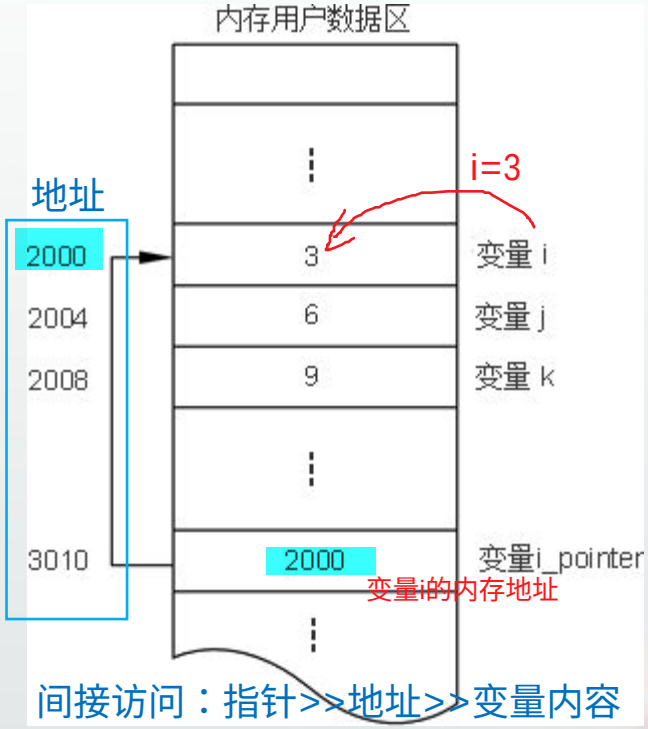
- 是指向另一种类型的复合类型，实现了对其他对象的间接访问。

- 声明和定义
- 类型 *指针变量名;

- int *ip1, *ip2; // both ip1 and ip2 are pointers to int
- double dp, *dp2; // dp2 is a pointer to double; dp is a double

内存单元与变量

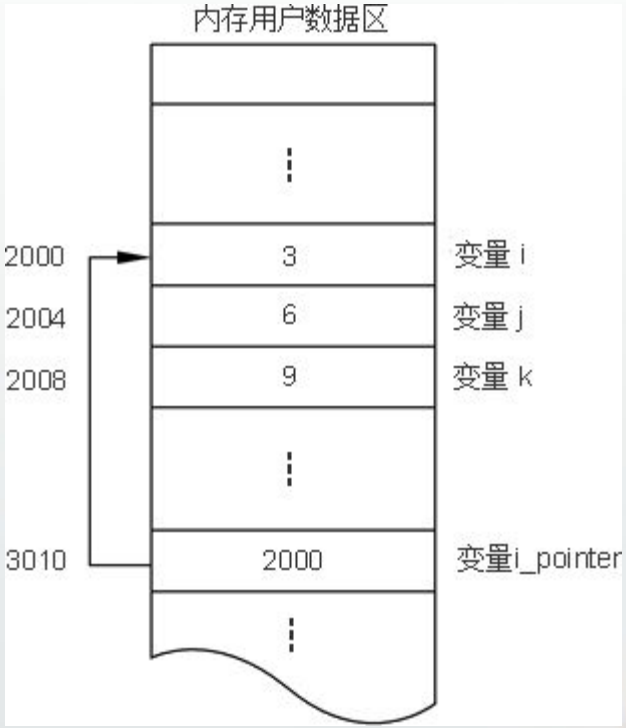
- 如果在程序中定义了一个变量，在编译时就给这个变量分配内存单元。
- 系统根据程序中定义的变量类型，分配一定长度的空间。
- 例如，C++编译系统一般为整型变量分配4个字节，为单精度浮点型变量分配4个字节，为字符型变量分配1个字节。内存区的每一个字节有一个编号，这图6.1就是“地址”。



数据存取访问方式

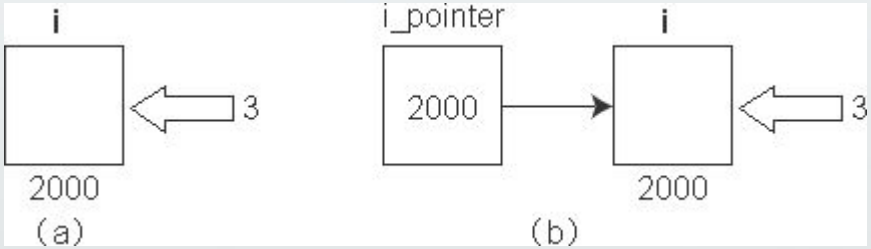
- 按变量地址存取内存中对应的变量值，所存取的值就是数据本身，称为直接存取方式，或直接访问方式。

- 按变量地址存取内存中对应的变量值，所存取的值就是数据所在的内存地址，再把变量值作为数据的内存地址来进行存取，称为间接存取(间接访问)的方式



直接访问和间接访问

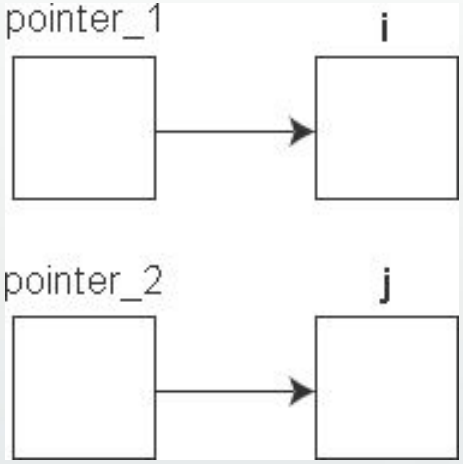
- 直接访问和间接访问的示意图。为了将数值3送到变量中，可以有两种方法：
  - 直接将数3送到整型变量i所标识的单元中。见图(a)。
  - 将3送到指针变量i_pointer所指向的单元(这就是变量i所标识的单元)中。见图(b)。



获取对象的地址

&var = 引用
=&var 递值

- int i, j; //定义整型变量i，j
- int *pointer_1, *pointer_2; //定义指针变量*pointer_1，*pointer_2



- pointer_1=&i; //将变量i的地址存放到指针变量pointer_1中
- pointer_2=&j; //将变量j的地址存放到指针变量pointer_2中
- //这样，pointer_1就指向了变量i，pointer_2就指向了变量j。

引用指针变量

- 有两个与指针变量有关的运算符：
  - &取地址运算符。
  - *指针运算符(或称间接访问运算符)。

- 例如：&a为变量a的地址，*p为指针变量p所指向的存储单元。

通过指针变量访问整型变量

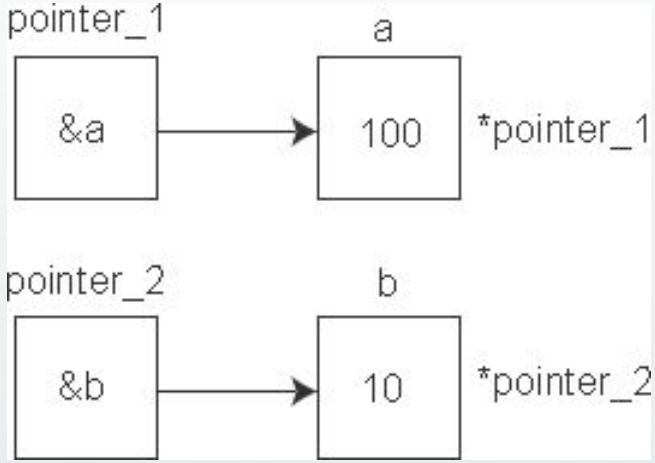
```
#include <iostream>
using namespace std;
int main( ){
  int a,b;  //定义整型变量a,b
  int *pointer_1,*pointer_2;  //定义指针变量*pointer_1,*pointer_2
  a=100;b=10;  //对a,b赋值
  pointer_1=&a;  //把变量a的地址赋给pointer_1
  pointer_2=&b;  //把变量a的地址赋给pointer_2
  cout<<a<<" "<<b<<endl;  //输出a和b的值
  cout<<*pointer_1<<" "<<*pointer_2<<endl;  //输出*pointer_1和
    *pointer_2的值
  return 0;
}
```

通过指针变量访问整型变量

运行结果为：

100 10　(a和b的值)

100 10　(*pointer_1和*pointer_2的值)



输入a和b两个整数，按先大后小的顺序输出a和b

```
#include <iostream>
using namespace std;
int main( ){
  int *p1,*p2,*p,a,b;
  cin>>a>>b; //输入两个整数
  p1=&a; //使p1指向a
  p2=&b; //使p2指向b
  if(a<b) {//如果a<b就使p1与p2的值交换
      p=p1;p1=p2;p2=p; //将p1的指向与p2的指向交换
  }
  cout<<"a="<<a<<" b="<<b<<endl;
  cout<<"max="<<*p1<<" min="<<*p2<<endl;
  return 0;
}
```

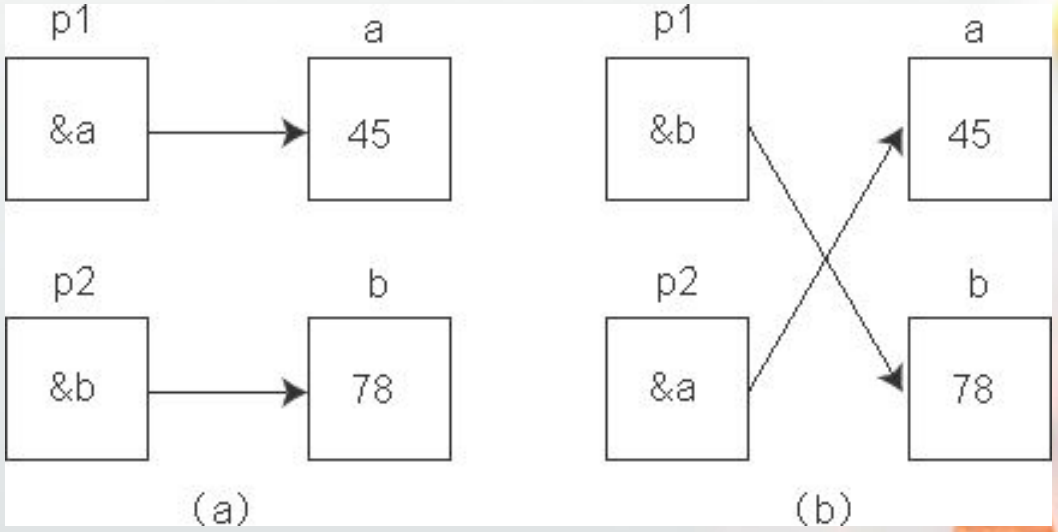
通过指针变量访问整型变量。

运行情况如下:

4578↙

a=45 b=78

max=78 min=45



指针值的4种状态

- 指向一个对象
- 指向紧邻对象所占空间的下一个位置
- 空指针
- 无效指针

2.4 const限定符

- 在定义变量时，如果加上关键字const，则变量的值在程序运行期间不能改变，这种变量称为常变量(constant variable)。

const+类型+赋值

- 在定义常变量时必须同时对它初始化（即指定其值），此后它的值不能再改变。

```
const int i = get_size();
const int j = 42;
```

不能写成：

```
const int j;
j=42;  //常变量不能被赋值
```

2.4.1 const的引用

```
const int bufSize = 512; // input buffer size
bufSize = 512; // error: attempt to write to const object

const int i = get_size(); // ok: initialized at run time
const int j = 42; // ok: initialized at compile time
const int k; // error: k is uninitialized const
```

```
int i = 42;
const int ci = i; // ok: the value in i is copied into ci
int j = ci; // ok: the value in ci is copied into j
```

2.4.1 const的引用

```
const int ci = 1024;
const int &r1 = ci;
// ok: both reference and underlying object are const
r1 = 42; // error: r1 is a reference to const
int &r2 = ci; // error: non const reference to a const object

int i = 42;
const int &r1 = i; // we can bind a const int& to a plain int object
const int &r2 = 42; // ok: r1 is a reference to const
const int &r3 = r1 * 2; // ok: r3 is a reference to const
int &r4 = r * 2; // error: r4 is a plain, non const reference
```

实验02.1：求圆或矩形的周长和面积

- 实验要求：
  - 输出提示"请输入选择：a.计算矩形；b.计算圆；q：退出”
  - 使用if/else语句根据输入的字符进行判断，提示并输入圆半径或矩形长宽，输出周长和面积，或退出循环
  - 在程序中使用const定义圆周率，使用while循环
  - 使用引用定义周长的别名
  - 使用指针定义面积

实验02.2：求裴波那契数列

- 实验要求：
  - 裴波那契数列的前几个数为1，1，2，3，5，8，……，其规律为：

    F1=1 (n=1)

    F2=1 (n=2)

    Fn=Fn-1+Fn-2 (n≥3)
  - 编写程序求此数列的前面40个数。
  - 使用for语句实现循环

内容

- 2.1 基本内置类型
- 2.2 变量
- 2.3 复合类型
- 2.4 const限定符

Q&A