东南大學 SCHOOL OF INTEGRATED CIRCUITS, SEU

集成电路学院

计算机科学基础I

——类型补充，函数

东南大学 集成电路学院 朱彬武

E-mail: bwzhu@seu.edu.cn

再谈数据类型

C++是有类型的语言

- C++语言的变量，必须在使用前定义，并且确定类型
- 编程语言可以根据是否有类型分为两类：
  - 以C++、JAVA为代表的语言强调类型，对类型的检查十分严格
  - 以Python，JavaScript为代表的语言不看重类型，甚至不需要事先定义
- 类型检查的好处在于可以尽早发现程序中的错误，但是会迫使程序员面对底层，而非事务逻辑

C++语言的类型

- 整型：bool、char、short、int、long、long long、unsigned int、.....
- 浮点型：float、double、long double
- 指针：int *、double *
- 自定义类型

不同类型之间的区别在于（1）所表达的数的范围：short < int < long （2）内存中所占据的大小：1字节到16字节（3）内存中的表达形式：补码、IEEE 756标准

整数的范围

- short ——2字节：$-2^{15}\sim 2^{15}-1$
- int ——4字节：$-2^{31}\sim 2^{31}-1$
- unsigned int ——4字节：$0\sim 2^{32}-1$

```
short i = 32768;
cout << i << endl;
// 10000000 00000000
short j = 32767 + 1;
cout << j << endl; //-32768
cout << j-1 << endl; //32767

unsigned short k = 32768;
cout << k << endl; //32768
```

一个新的运算符——sizeof

- sizeof是一个运算符，给出某个类型或变量在内存中所占据的字节数

```
cout << sizeof(bool) << endl; //1
cout << sizeof(short) << endl; //2
cout << sizeof(int) << endl;  //4

float a = 2.0;
double b = 1.0;
cout << sizeof(a) << endl; //4
cout << sizeof(b) << endl; //8
```

浮点运算的精度

- 不是每个小数都能在计算中精确表示：0.1，0.2，0.3，0.4，……

```
double i = 0.1, j = 0.2;
double s = i + j;

if (s == 0.3)
    cout << "相等" << endl;
else
    cout << "不相等" << endl;

cout << setprecision(17) << s << endl;
cout << setprecision(17) << 0.3 << endl;
```

0.30000000000000004

0.29999999999999999

例子：

$(0.4)_{10}$

2×0.4 = 0.8 ……0

2×0.8 = 1.6 ……1

2×0.6 = 1.2 ……1

2×0.2 = 0.4 ……0

2×0.4 = 0.8 ……0

……

$(0.4)_{10} \approx (0.0110)_2$（保留4位）

浮点运算的精度


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

double i = 0.1, j = 0.2;
double s = i + j;
double epsilon = 1e-12

if(fabs(s-0.3)<epsilon)
    cout << "相等" << endl;
else
    cout << "不相等" << endl;
```

- 引入一个极小量epsilon作为判断标准，忽略足够小的浮点数误差问题
- fabs是一个求浮点数的绝对值函数，定义在头文件<cmath>中，因此要在开头加上#include<cmath>

字符类型

字符类型char

- char 是一种字符类型，表示单个ASCII字符：'a'，'1'，字符字面量必须用单引号包起来

```
char c1 = 'S';
char c2 = 'E';
char c3 = 'U';

cout << c1 << c2 << c3 << endl;
```

- char本质上是一种整型，使用1个字节，存储的是对应字符的 ASCII 码数值

```
char c1, c2;
c1 = '1';
c2 = 49;
if(c1==c2){
    cout << "相等" << endl;
}
else{
    cout << "不相等" << endl;
}
cout << c1 << endl;
cout << c2 << endl;
```

cout 输出 char 时会按字符而不是整数打印

ASCII字符表

ASCII可显示字符（共95个）

二进制	十进制	十六进制	图形	二进制	十进制	十六进制	图形	二进制	十进制	十六进制	图形
0010 0000	32	20	(space)	0100 0000	64	40	@	0110 0000	96	60	`
0010 0001	33	21	!	0100 0001	65	41	A	0110 0001	97	61	a
0010 0010	34	22	"	0100 0010	66	42	B	0110 0010	98	62	b
0010 0011	35	23	#	0100 0011	67	43	C	0110 0011	99	63	c
0010 0100	36	24	$	0100 0100	68	44	D	0110 0100	100	64	d
0010 0101	37	25	%	0100 0101	69	45	E	0110 0101	101	65	e
0010 0110	38	26	&	0100 0110	70	46	F	0110 0110	102	66	f
0010 0111	39	27	'	0100 0111	71	47	G	0110 0111	103	67	g
0010 1000	40	28	(	0100 1000	72	48	H	0110 1000	104	68	h
0010 1001	41	29	)	0100 1001	73	49	I	0110 1001	105	69	i
0010 1010	42	2A	*	0100 1010	74	4A	J	0110 1010	106	6A	j
0010 1011	43	2B	+	0100 1011	75	4B	K	0110 1011	107	6B	k
0010 1100	44	2C	,	0100 1100	76	4C	L	0110 1100	108	6C	l
0010 1101	45	2D	-	0100 1101	77	4D	M	0110 1101	109	6D	m
0010 1110	46	2E	.	0100 1110	78	4E	N	0110 1110	110	6E	n
0010 1111	47	2F	/	0100 1111	79	4F	O	0110 1111	111	6F	o
0011 0000	48	30	0	0101 0000	80	50	P	0111 0000	112	70	p
0011 0001	49	31	1	0101 0001	81	51	Q	0111 0001	113	71	q
0011 0010	50	32	2	0101 0010	82	52	R	0111 0010	114	72	r
0011 0011	51	33	3	0101 0011	83	53	S	0111 0011	115	73	s
0011 0100	52	34	4	0101 0100	84	54	T	0111 0100	116	74	t
0011 0101	53	35	5	0101 0101	85	55	U	0111 0101	117	75	u
0011 0110	54	36	6	0101 0110	86	56	V	0111 0110	118	76	v
0011 0111	55	37	7	0101 0111	87	57	W	0111 0111	119	77	w
0011 1000	56	38	8	0101 1000	88	58	X	0111 1000	120	78	x
0011 1001	57	39	9	0101 1001	89	59	Y	0111 1001	121	79	y
0011 1010	58	3A	:	0101 1010	90	5A	Z	0111 1010	122	7A	z
0011 1011	59	3B	;	0101 1011	91	5B	[	0111 1011	123	7B	{
0011 1100	60	3C	<	0101 1100	92	5C	\	0111 1100	124	7C	|
0011 1101	61	3D	=	0101 1101	93	5D	]	0111 1101	125	7D	}
0011 1110	62	3E	>	0101 1110	94	5E	^	0111 1110	126	7E	~
0011 1111	63	3F	?	0101 1111	95	5F	_				

字符计算


```
char c = 'A';
c++;
cout << c << endl;

char t = 'B' + 'a' - 'A';
cout << t << endl;
```

- 因为char本质上是整型，因此可以和整型做计算
- 大写字母 + 'a'-'A'可以把一个大写字母变成小写字母
- 小写字母 - ('a'-'A')可以把一个小写字母变成大写字母

转义字符

- 转义字符用来表达无法打印出来的控制字符或特殊字符，由一个反斜杠“\”开头，后面跟上另一个字符，这两个字符合起来，组成了一个字符

```
cout << "请分别输入包裹的英镑和盎司，如输入\"8 5\"表示8英镑5盎司:" << endl;

cout << "请分别输入包裹的英镑和盎司，如输入"8 5"表示8英镑5盎司:" << endl; //error
```

- 常用转义字符

转义字符	含义
\n	换行（newline）
\t	水平制表符（tab，约 4 个空格）
\\	一个反斜杠
\'	单引号
\"	双引号
\r	回车（return）
\b	退格（backspace）
\0	字符串结束标志（空字符）

转义字符实例


```
cout << "123\'456\'\n789\b0";

cout <<"12\t456" << endl;
cout << "123\t456" << endl;

cout << "123\n456\r7" << endl;
```

输出

```
123'456'
780

12      456
123     456

123
756
```

- \b (Backspace): 退格，光标要回到前面的位置上去
- \t (Tab)：\t使得光标从下一个制表位开始，用\t可以使得上下两行对齐
- \r和\n：回车和换行，来源于打字机的动作，回车使得光标移动到当前行的开头，换行使光标移动到下一行的开头

类型转换

自动类型转换

- $(3)_{10} + (11)_8 = ?$
- 当运算符的两边出现不一致的类型时，会自动转换成较大的类型
  - 大的意思是能表达的数的范围更大
  - char -> short -> int -> long -> long long
  - int -> float -> double

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

int main(){
    int pound, ounce;
    cout << "请分别输入包裹的英镑和盎司，如输入\"8 5\"表示8英镑5盎司:" << endl;
    cin >> pound >> ounce;

    cout << "质量是" << (pound + ounce / 16.0) * 0.453592 << "千克"  << endl;
}
```

强制类型转换


```
int a = 10;
int b = 2;

cout << pow((double)a, b) << endl;
```

- 有时候我们需要把一个量强制转换成另一个类型，需要：（类型）值
- 例子：
  - (int)3.2
  - (short)32

强制类型转换

- 注意存在安全性的问题，小的变量并不总能表达大的量
  - (short)32768 == -32768

$$32768 = (10000000\ 00000000)_2$$

  - (char)32768 == '\0'
- 注意：强制类型转换，只是计算出了一个新的类型的值，并不改变变量本身，无论是变量的类型还是值都不会改变

```
int i = 32768;
short s = (short)i;
cout << s << endl;  // -32768
cout << i << endl;  // 32768
```

强制类型转换的优先级

强制类型转换的优先级高于四则运算

```
double a = 4.0;
double b = 3.0;
int i = (int)a / b;
int j = (int)(a / b);
```

函数的定义、调用与返回

求幂

求2的7次方、5的6次方、7的5次方（不用<cmath>提供的pow函数）

```
int res1 = 1;
for(i=1; i<=7; i++){
    res1 *= 2;
}
cout << res1 << endl;
```

```
int res2 = 1;
for(i=1; i<=6; i++){
    res2 *= 5;
}
cout << res2 << endl;
```

```
int res3 = 1;
for(i=1; i<=5; i++){
    res3 *= 7;
}
cout << res3 << endl;
```

- 三段几乎一模一样的代码
- 当我们要复制自己写过的代码时，一定是代码质量出现了问题。

引入变量的动机

在学习变量之前：

```
std::cout << 23489 + 12345 << std::endl;
std::cout << 23489 - 12345 << std::endl;
std::cout << 23489 * 12345 << std::endl;
std::cout << 23489 / 12345 << std::endl;
std::cout << 23489 % 12345 << std::endl;
```

- 需要多次输入两个很长的数
- 换两个其他的数字计算的时候需要手动修改每一个地方

引入变量之后：

```
int a = 23489;
int b = 12345;
std::cout << a + b << std::endl;
std::cout << a - b << std::endl;
std::cout << a * b << std::endl;
std::cout << a / b << std::endl;
std::cout << a % b << std::endl;
```

- 定义一次，多次使用
- 换两个其他的数字计算的时候只需要修改a和b的值

引入函数的动机

我们常常需要重复做一类操作。与其每次都重新写，不如把这段操作封装成一个函数。以后只要“调用”它，就能完成同样的事情。

- 函数（function）是一块代码，接收零个或多个参数，做一件事情，并返回零个或一个值
- 可以想象数学中的函数：$y = f(x) = |x|$

求幂——引入函数


```
int pow(int base, int exponent){
    int res = 1;
    for(int i = 1; i <= exponent; i++){
        res *= base;
    }
    return res;
}

int main(){
    int res1 = pow(2, 7);
    cout << res1 << endl;

    int res2 = pow(5, 6);
    cout << res2 << endl;

    int res3 = pow(7, 5);
    cout << res3 << endl;
    return 0;
}
```

引入函数之后，我们的代码结构更加清晰

函数定义

函数名

参数表

函数头

返回类型

```
int pow(int base, int exponent){
    int res = 1;
    for(int i = 1; i <= exponent; i++){
        res *= base;
    }
    return res;
}
```

函数体

函数调用（1）

- 要用函数，就需要去调用函数
- 函数调用方式：函数名（参数值）
  - 即使没有参数也需要后面这对括号（）
- 如果有参数，则需要给出正确的数量和顺序

```
int res1 = pow(2, 7);
```

```
void cheer(){
    cout << "Cheer Cheer Cheer" << endl;
}

int main(){
    cheer();
}
```

函数调用（2）

- 传入的参数值会被按照顺序依次用来初始化函数中的参数
- 参数在函数里就像变量一样被使用

```
int res1 = pow(2, 7);

int pow(int base, int exponent){
    int res = 1;
    for(int i = 1; i <= exponent; i++){
        res *= base;
    }
    return res;
}
```

函数返回值


```
int pow(int base, int exponent){
    int res = 1;
    for(int i = 1; i <= exponent; i++){
        res *= base;
    }
    return res;
}
```

- 函数在执行完成之后，大部分时候要“交还”一个返回值
- 返回值传到调用该函数的地方，返回值类型在函数定义时指定
- 通过return实现，return的作用就是停止函数的执行，然后返回一个值：return 表达式;

函数返回值


```
int res1 = pow(2, 7);
int res2 = pow(pow(2,2), 6);
pow(pow(2,2), 6)
```

- 可以赋值给变量
- 可以再传递给函数
- 甚至可以丢弃，不影响编译

main函数中的return 0


```
int main(){

    cout << "Hello World!" << endl;
    return 0;

}
```

- main函数是程序的入口函数，它的返回值会传给操作系统
- return 0表示程序成功正常结束，非 0 的返回值通常表示某种错误或异常情况。
- 在 C++11 及之后的标准里：如果 main 函数的结尾没有写 return 0;，编译器会自动补上 return 0; 但是仍然建议大家自己写，养成良好的习惯

函数也可以没有返回值


```
void cheer(){
    cout << "Cheer Cheer Cheer" << endl;
}
```

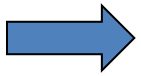
```
void cheer(){
    cout << "Cheer Cheer Cheer" << endl;
    return;
}
```

- 函数可以没有返回值，return语句可以没有，此时函数的返回值类型是void
- 调用的时候不能做返回值的赋值

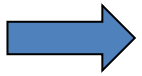
函数也可以有多个return

请编写一个函数，这个函数接收两个整数参数 a 和 b，并返回其中较大的那个数。

```
int max(int a, int b){
    int ret;
    if(a > b){
        ret = a;
    }
    else{
        ret = b;
    }
    return ret;
}
```



```
int max(int a, int b){
    if(a > b){
        return a;
    }
    else{
        return b;
    }
}
```



```
int max(int a, int b){
    if(a > b){
        return a;
    }
    return b;
}
```

但是会更加推荐大家采用第一种写法，符合“单一出口”的设计理念：容易阅读、调试、维护