C语言

第1讲：数据类型

赵岩

哈尔滨工业大学软件学院

August 20, 2011

目录

目录

集成开发环境IDE

- 主要产品
  - Visual Studio
  - code block
- 开发流程
  1. 建立项目
  2. 编辑文件
  3. 编译，运行
  4. 调试程序

符号分类

- 关键字（Keyword）
  - 又称为保留字，C语言中预先规定的具有固定含义的一些单词(`if`, `while`, ...)。详见附录B
- 标识符（Identifier）
  - 系统预定义标识符: main，printf
  - 用户自定义标识符: 自定义变量名，函数名和类型名
- 运算符（Operator）
  - 34种，(+, -, *, ...)。详见附录D
- 分隔符（Separator）
  - 空格、回车/换行、逗号等
- 其它符号
  - 大花括号“{”和“}”通常用于标识函数体或者一个语句块
  - “/*”和“*/”是程序注释所需的定界符

符号分类

- 关键字（Keyword）
  - 又称为保留字，C语言中预先规定的具有固定含义的一些单词(`if`, `while`, ...)。详见附录B
- 标识符（Identifier）
  - 系统预定义标识符: main，printf
  - 用户自定义标识符: 自定义变量名，函数名和类型名
- 运算符（Operator）
  - 34种，(+, -, *, ...)。详见附录D
- 分隔符（Separator）
  - 空格、回车/换行、逗号等
- 其它符号
  - 大花括号“{”和“}”通常用于标识函数体或者一个语句块
  - “/*”和“*/”是程序注释所需的定界符

符号分类

- 关键字（Keyword）
  - 又称为保留字，C语言中预先规定的具有固定含义的一些单词(`if`, `while`, ...)。详见附录B
- 标识符（Identifier）
  - 系统预定义标识符: main，printf
  - 用户自定义标识符: 自定义变量名，函数名和类型名
- 运算符（Operator）
  - 34种，(+, -, *, ...)。详见附录D
- 分隔符（Separator）
  - 空格、回车/换行、逗号等
- 其它符号
  - 大花括号“{”和“}”通常用于标识函数体或者一个语句块
  - “/*”和“*/”是程序注释所需的定界符

符号分类

- 关键字（Keyword）
  - 又称为保留字，C语言中预先规定的具有固定含义的一些单词(`if`, `while`, ...)。详见附录B
- 标识符（Identifier）
  - 系统预定义标识符: main，printf
  - 用户自定义标识符: 自定义变量名，函数名和类型名
- 运算符（Operator）
  - 34种，(+, -, *, ...)。详见附录D
- 分隔符（Separator）
  - 空格、回车/换行、逗号等
- 其它符号
  - 大花括号“{”和“}”通常用于标识函数体或者一个语句块
  - “/*”和“*/”是程序注释所需的定界符

符号分类

- 关键字（Keyword）
  - 又称为保留字，C语言中预先规定的具有固定含义的一些单词(`if`, `while`, ...)。详见附录B
- 标识符（Identifier）
  - 系统预定义标识符: main，printf
  - 用户自定义标识符: 自定义变量名，函数名和类型名
- 运算符（Operator）
  - 34种，(+, -, *, ...)。详见附录D
- 分隔符（Separator）
  - 空格、回车/换行、逗号等
- 其它符号
  - 大花括号“{”和“}”通常用于标识函数体或者一个语句块
  - “/*”和“*/”是程序注释所需的定界符

标识符例子

Demo 1: Identifiers example

```
1  #include<stdio.h>
2  int Add(int a,int b){
3      return (a + b);
4  }
5  main(){
6      int x,y,sum = 0;
7      printf("Input two integers:");
8      scanf("%d,%d",&x,&y);
9      sum = Add(x,y);
10     printf("sum=%d\n",sum);
11 }
```

自定义标示符命名规则

- 关键字（keyword）不可作为标识符
  - `int`, `float`, `for`, `while`, `if`等
- 尽量避开系统预定义标示符：printf, main
- 以字母或下划线开头，后跟字母、数字、下划线组成的串
- **不可以是数字开头。**
- 大小写敏感：name, Name, NAME是三个不同的标示符
  - 一般大写字母标示符在C语言中代表常量

自定义标示符命名规则

- 关键字（keyword）不可作为标识符
  - `int`, `float`, `for`, `while`, `if`等
- 尽量避开系统预定义标示符：printf, main
- 以字母或下划线开头，后跟字母、数字、下划线组成的串
- 不可以是数字开头。
- 大小写敏感：name, Name, NAME是三个不同的标示符
  - 一般大写字母标示符在C语言中代表常量

自定义标示符命名规则

- 关键字（keyword）不可作为标识符
  - `int`, `float`, `for`, `while`, `if`等
- 尽量避开系统预定义标示符：printf, main
- 以字母或下划线开头，后跟字母、数字、下划线组成的串
- 不可以是数字开头。
- 大小写敏感：name, Name, NAME是三个不同的标示符
  - 一般大写字母标示符在C语言中代表常量

自定义标示符命名规则

- 关键字（keyword）不可作为标识符
  - `int`, `float`, `for`, `while`, `if`等
- 尽量避开系统预定义标示符：printf, main
- 以字母或下划线开头，后跟字母、数字、下划线组成的串
- **不可以是数字开头。**
- 大小写敏感：name, Name, NAME是三个不同的标示符
  - 一般大写字母标示符在C语言中代表常量

自定义标示符命名规则

- 关键字（keyword）不可作为标识符
  - `int`, `float`, `for`, `while`, `if`等
- 尽量避开系统预定义标示符：printf, main
- 以字母或下划线开头，后跟字母、数字、下划线组成的串
- **不可以是数字开头。**
- 大小写敏感：name, Name, NAME是三个不同的标示符
  - 一般大写字母标示符在C语言中代表常量

自定义标示符命名习惯

- 标识符要直观，能表达它的功能
  - 英语还是拼音？
- 下划线和大小写通常用来增强可读性
  - variablename (bad name)
  - VARIABLE_NAME (常量名)
  - variable_name, variableName (good name)
- 某些功能的变量采用习惯命名
  - 如：循环语句所采用的循环变量习惯用i, j, k

自定义标示符命名习惯

- 标识符要直观，能表达它的功能
  - 英语还是拼音？
- 下划线和大小写通常用来增强可读性
  - variablename (bad name)
  - VARIABLE_NAME (常量名)
  - variable_name, variableName (good name)
- 某些功能的变量采用习惯命名
  - 如：循环语句所采用的循环变量习惯用i, j, k

自定义标示符命名习惯

- 标识符要直观，能表达它的功能
  - 英语还是拼音？
- 下划线和大小写通常用来增强可读性
  - variablename (bad name)
  - VARIABLE_NAME (常量名)
  - variable_name, variableName (good name)
- 某些功能的变量采用习惯命名
  - 如：循环语句所采用的循环变量习惯用i, j, k

数据的类型

- 数据为什么要区分类型？
  - Untyped Perl。

Demo 2: A simple perl example

```
1 #!/usr/bin/perl
2 my $var = 1, $var1, $var2;
3 $var = $var + 2.3;
4 print $var,"\n";
5
6 $var1 = $var."abc";
7 print $var1,"\n";
8 $var2 = $var1+"abc";
9 print $var2,"\n";
```

区分数据类型的优点

- 数据的类型代表不同的
  - 数据表示形式
  - 合法的取值范围
  - 占用内存空间大小
  - 可参与的运算种类
- 牺牲灵活性，增加可靠性。避免类型方面的错误

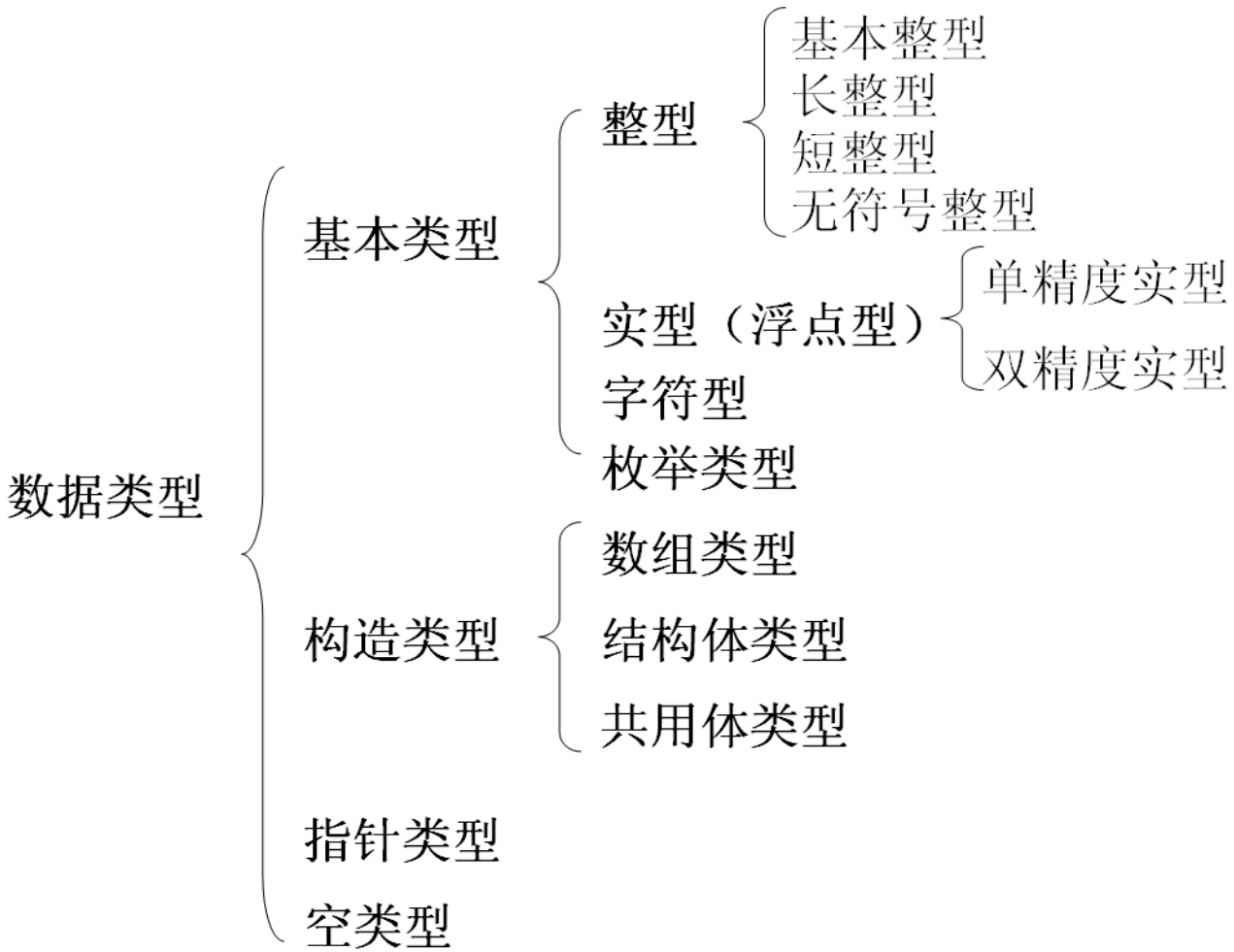
区分数据类型的优点

- 数据的类型代表不同的
  - 数据表示形式
  - 合法的取值范围
  - 占用内存空间大小
  - 可参与的运算种类
- 牺牲灵活性，增加可靠性。避免类型方面的错误

内存的度量

英文	中文	换算
bit	位	
Byte	字节	1B==8bit
Kilobyte(KB)	K	1KB==1024B
Megabyte(MB)	兆	1MB==1024KB
Gigabyte(GB)	吉	1GB==1024MB
Terabyte(TB)	T	1TB==1024GB

数据类型分类


数据类型
基本类型
整型
基本整型
长整型
短整型
无符号整型
实型（浮点型）
单精度实型
双精度实型
字符型
枚举类型
构造类型
数组类型
结构体类型
共用体类型
指针类型
空类型

基本数据类型

- int
  - 整数，在目前绝大多数机器上占4个字节
- float
  - 单精度浮点数，一般是4个字节
- double
  - 双精度浮点数，一般是8个字节
- char
  - 字符，一般是1个字节长
  - 用来表示256个ASCII字符，或者0-255或者(-128-127)的整数

基本数据类型

- int
  - 整数，在目前绝大多数机器上占4个字节
- float
  - 单精度浮点数，一般是4个字节
- double
  - 双精度浮点数，一般是8个字节
- char
  - 字符，一般是1个字节长
  - 用来表示256个ASCII字符，或者0-255或者(-128-127)的整数

基本数据类型

- int
  - 整数，在目前绝大多数机器上占4个字节
- float
  - 单精度浮点数，一般是4个字节
- double
  - 双精度浮点数，一般是8个字节
- char
  - 字符，一般是1个字节长
  - 用来表示256个ASCII字符，或者0-255或者(-128-127)的整数

基本数据类型

- int
  - 整数，在目前绝大多数机器上占4个字节
- float
  - 单精度浮点数，一般是4个字节
- double
  - 双精度浮点数，一般是8个字节
- char
  - 字符，一般是1个字节长
  - 用来表示256个ASCII字符，或者0-255或者(-128-127)的整数

修饰符

- short
  - short int，短整数，一般占2字节。通常简写为short
- long
  - long int，长整数，一般占4字节。通常简写为long
  - long double，高精度浮点数，一般占12字节。
- signed
  - 修饰char、int、short、long，说明他们是有符号的整数（正整数、0和负整数）。缺省都是有符号的，所以这个修饰符常省略
- unsigned
  - 修饰char、int、short、long，说明他们是无符号的整数（正整数和0）

修饰符

- short
  - short int，短整数，一般占2字节。通常简写为short
- long
  - long int，长整数，一般占4字节。通常简写为long
  - long double，高精度浮点数，一般占12字节。
- signed
  - 修饰char、int、short、long，说明他们是有符号的整数（正整数、0和负整数）。缺省都是有符号的，所以这个修饰符常省略
- unsigned
  - 修饰char、int、short、long，说明他们是无符号的整数（正整数和0）

修饰符

- short
  - short int，短整数，一般占2字节。通常简写为short
- long
  - long int，长整数，一般占4字节。通常简写为long
  - long double，高精度浮点数，一般占12字节。
- signed
  - 修饰char、int、short、long，说明他们是有符号的整数（正整数、0和负整数）。缺省都是有符号的，所以这个修饰符常省略
- unsigned
  - 修饰char、int、short、long，说明他们是无符号的整数（正整数和0）

修饰符

- short
  - short int，短整数，一般占2字节。通常简写为short
- long
  - long int，长整数，一般占4字节。通常简写为long
  - long double，高精度浮点数，一般占12字节。
- signed
  - 修饰char、int、short、long，说明他们是有符号的整数（正整数、0和负整数）。缺省都是有符号的，所以这个修饰符常省略
- unsigned
  - 修饰char、int、short、long，说明他们是无符号的整数（正整数和0）

数据类型的长度

- 不要对变量所占的字节数想当然
- 用sizeof获得变量或者数据类型的长度
- 它是一个C语言的关键字，并不是函数
- 可以用两种形式使用
  - sizeof(变量名)
  - sizeof(类型)
- 返回结果为一个size_t类型的数
  - typedef **unsigned int** size_t
- 求出的结果为类型占用的字节数

数据类型的长度

- 不要对变量所占的字节数想当然
- 用`sizeof`获得变量或者数据类型的长度
- 它是一个C语言的关键字，并不是函数
- 可以用两种形式使用
  - `sizeof`(变量名)
  - `sizeof`(类型)
- 返回结果为一个size_t类型的数
  - typedef **unsigned int** size_t
- 求出的结果为类型占用的字节数

数据类型的长度

- 不要对变量所占的字节数想当然
- 用sizeof获得变量或者数据类型的长度
- 它是一个C语言的关键字，并不是函数
- 可以用两种形式使用
  - sizeof(变量名)
  - sizeof(类型)
- 返回结果为一个size_t类型的数
  - typedef **unsigned int** size_t
- 求出的结果为类型占用的字节数

数据类型的长度

- 不要对变量所占的字节数想当然
- 用sizeof获得变量或者数据类型的长度
- 它是一个C语言的关键字，并不是函数
- 可以用两种形式使用
  - sizeof(变量名)
  - sizeof(类型)
- 返回结果为一个size_t类型的数
  - typedef **unsigned int** size_t
- 求出的结果为类型占用的字节数

数据类型的长度

- 不要对变量所占的字节数想当然
- 用sizeof获得变量或者数据类型的长度
- 它是一个C语言的关键字，并不是函数
- 可以用两种形式使用
  - sizeof(变量名)
  - sizeof(类型)
- 返回结果为一个size_t类型的数
  - typedef **unsigned int** size_t
- 求出的结果为类型占用的字节数

数据类型的长度

- 不要对变量所占的字节数想当然
- 用sizeof获得变量或者数据类型的长度
- 它是一个C语言的关键字，并不是函数
- 可以用两种形式使用
  - sizeof(变量名)
  - sizeof(类型)
- 返回结果为一个size_t类型的数
  - typedef **unsigned int** size_t
- 求出的结果为类型占用的字节数

Demo 3: A simple example about data type

```
1     printf("Data type     Number of bytes\n");
2     printf("------------ ----------------\n");
3     printf("char          %d\n", sizeof(char));
4     printf("int           %d\n", sizeof(int));
5     printf("short int     %d\n", sizeof(short));
6     printf("long int      %d\n", sizeof(long));
7     printf("float         %d\n", sizeof(float));
8     printf("double        %d\n", sizeof(double));
```

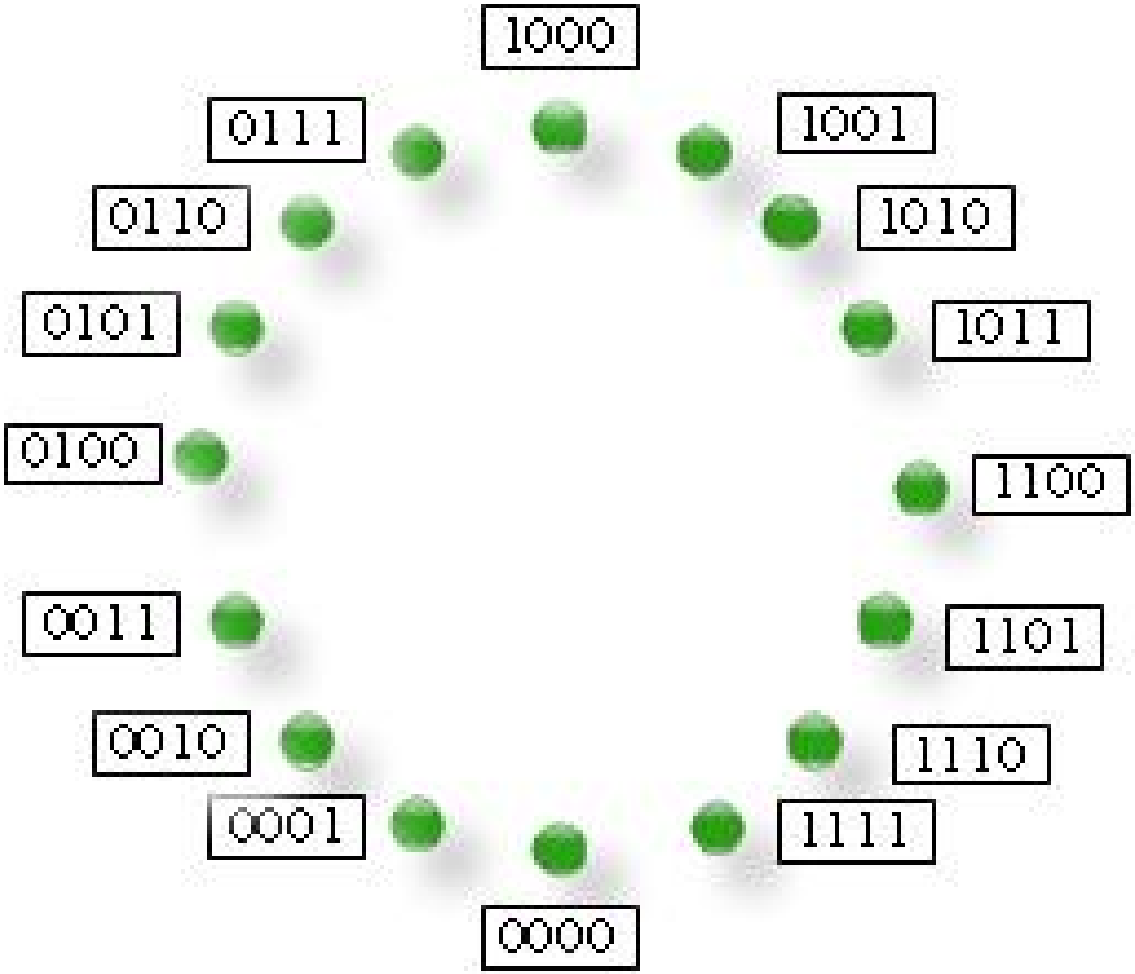
无符号数的运算陷阱

```
1 char str1[] = "abcdefg";
2 char str2[] = "abc"
3 if(strlen(str2)-strlen(str1)>0)
4 {
5     printf("str2 is longer than str1\n");
6 }
```

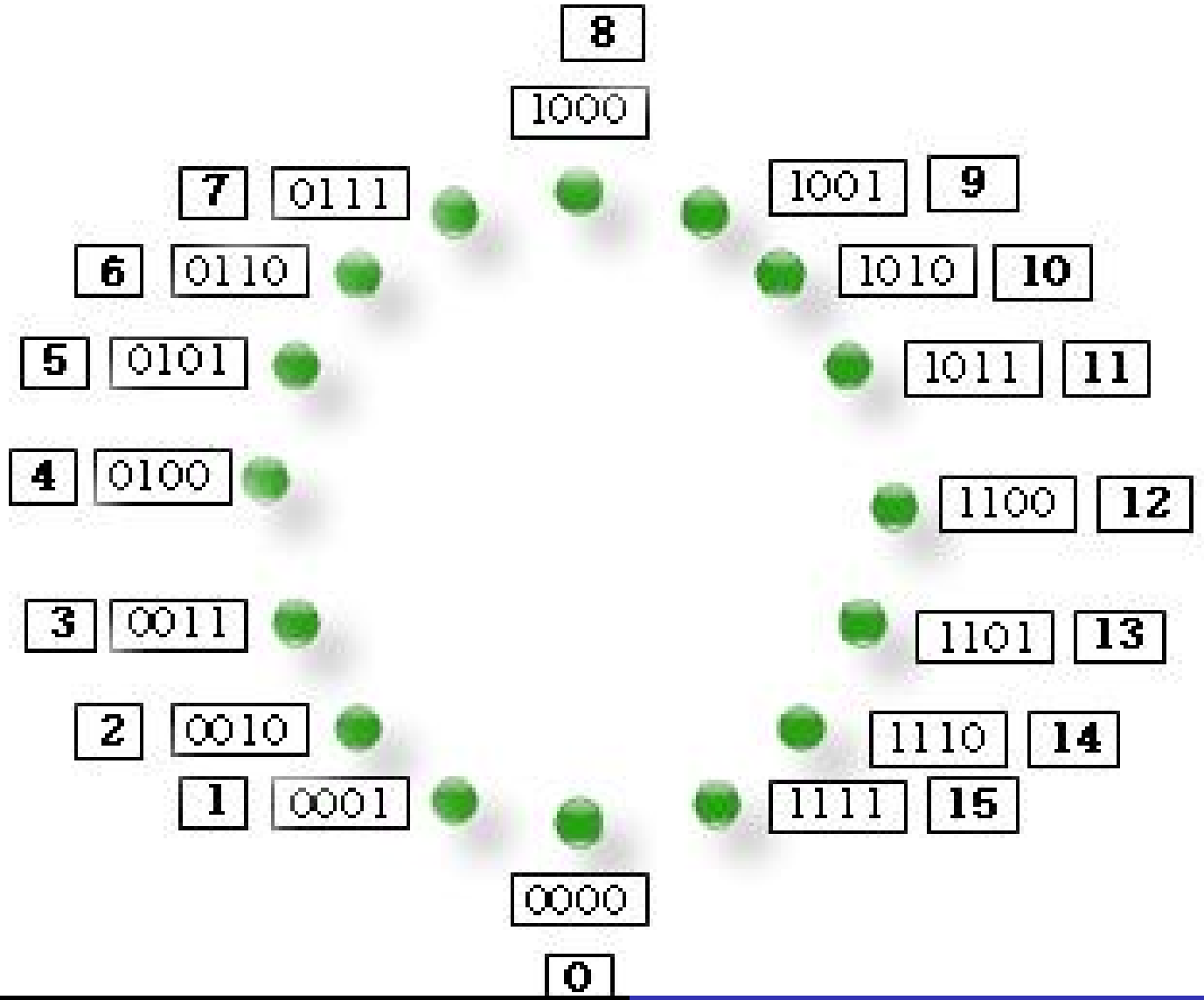
整形数据的溢出

- 任何类型都有取值范围。超过此范围的数值，就会产生数值**溢出**，得到一个不正确的结果

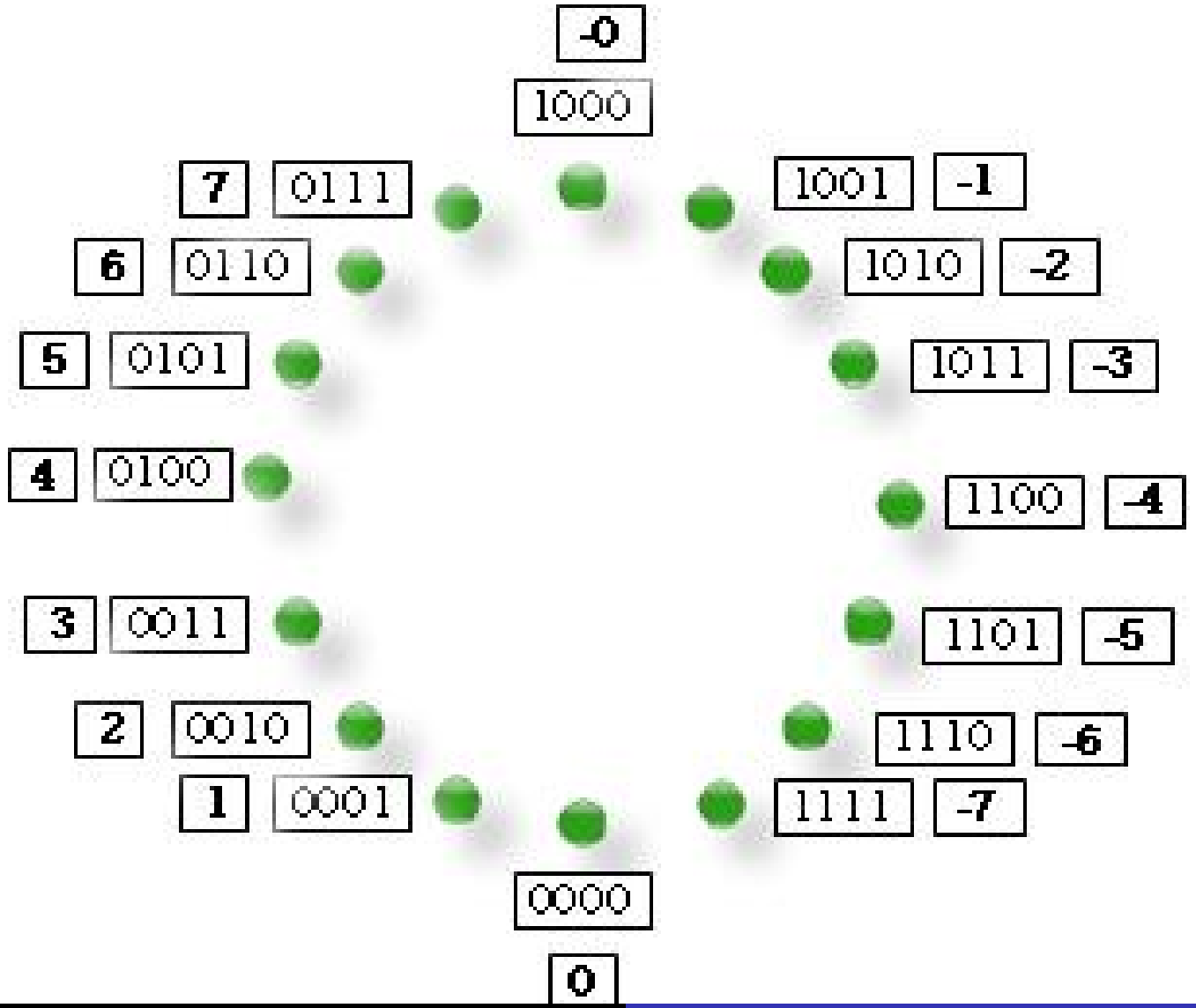
二进制表示


1000
1001
1010
1011
1100
1101
1110
1111
0000
0001
0010
0011
0100
0101
0110
0111

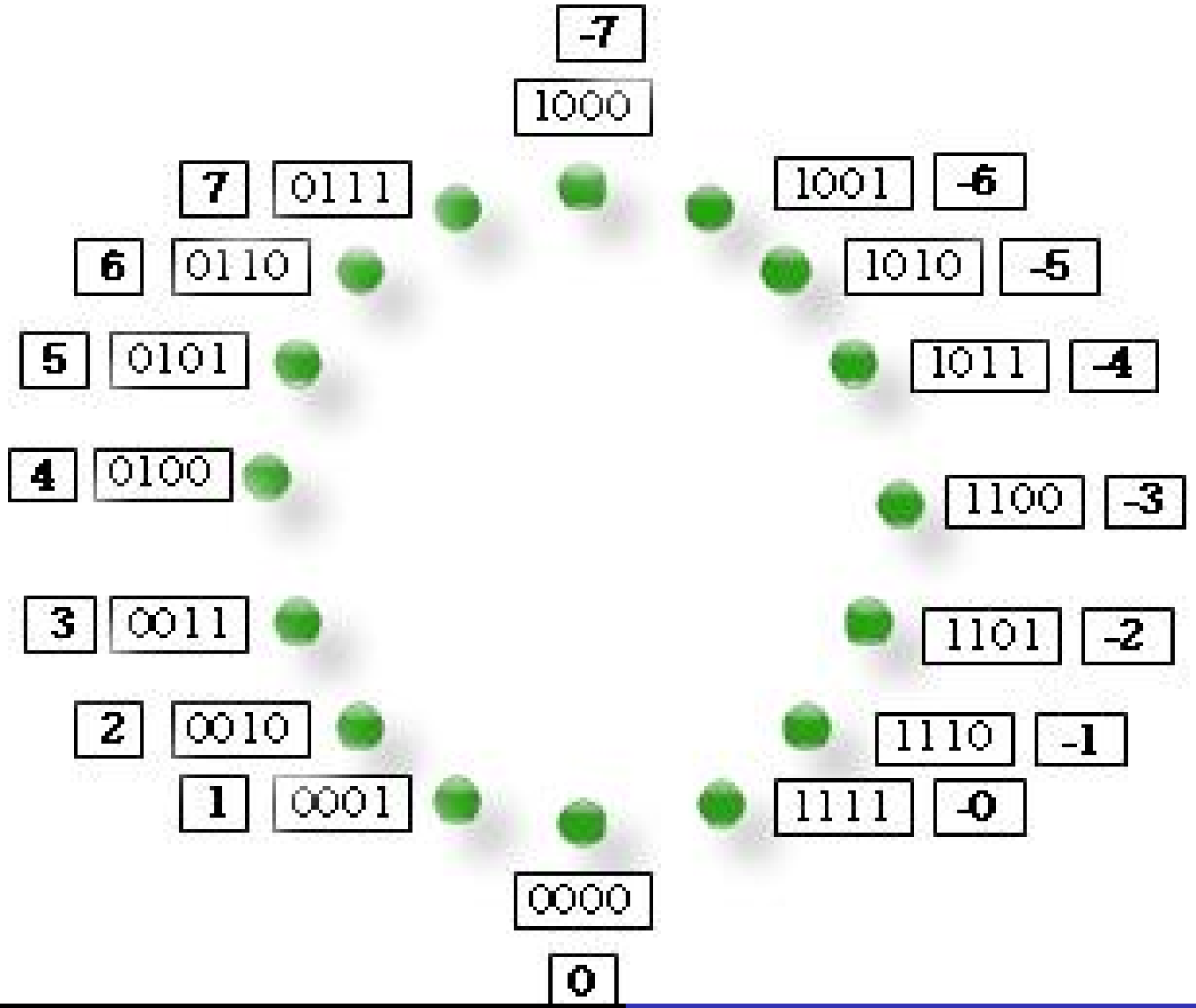
无符号数表示


8
1000
7
0111
1001
9
6
0110
1010
10
5
0101
1011
11
4
0100
1100
12
3
0011
1101
13
2
0010
1110
14
1
0001
1111
15
0000
0

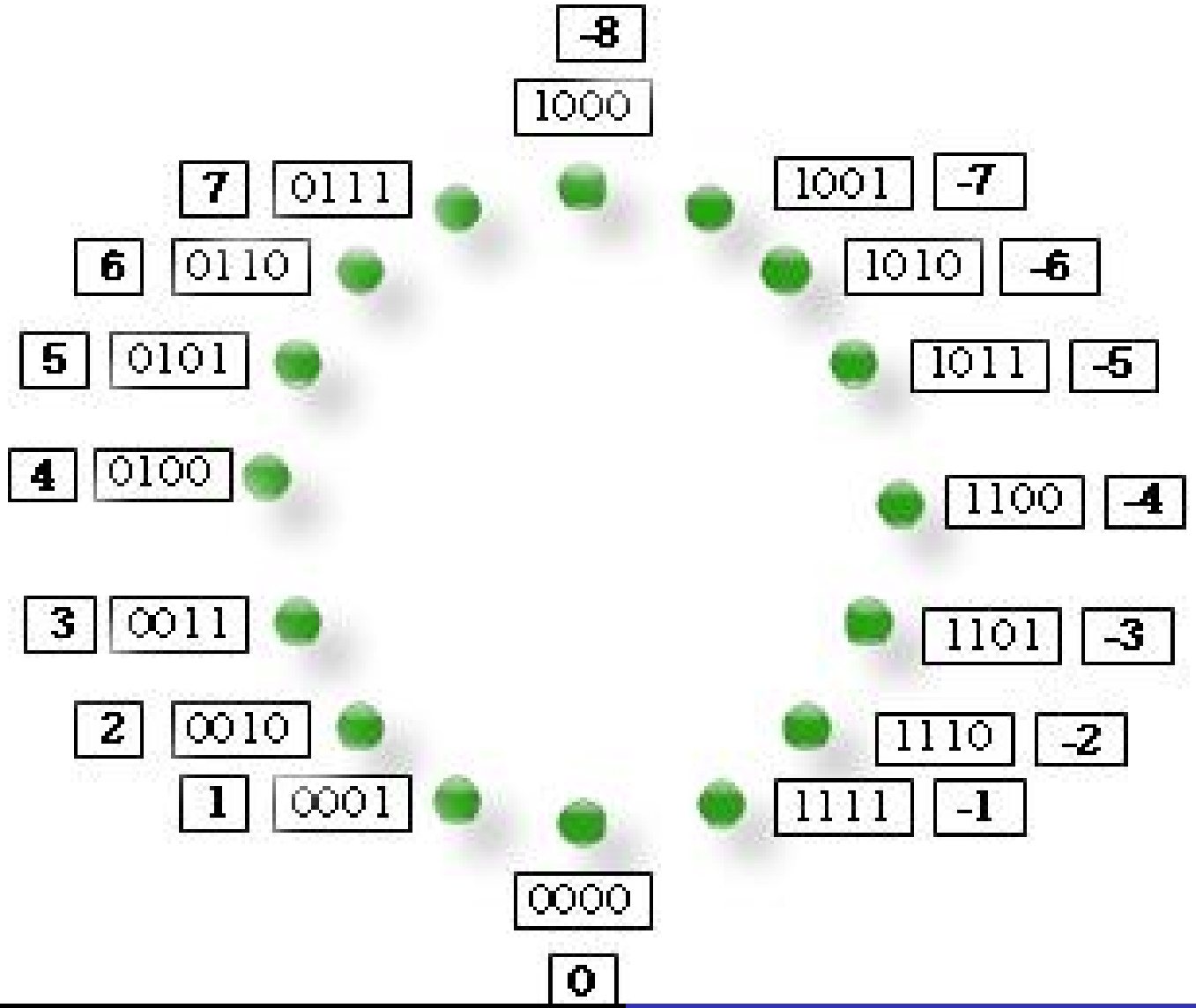
原码


-0
1000
7
0111
1001
-1
6
0110
1010
-2
5
0101
1011
-3
4
0100
1100
-4
3
0011
1101
-5
2
0010
1110
-6
1
0001
1111
-7
0000
0

反码


-7
1000
7
0111
1001
-6
6
0110
1010
-5
5
0101
1011
-4
4
0100
1100
-3
3
0011
1101
-2
2
0010
1110
-1
1
0001
1111
-0
0000
0

补码


-8
1000
7
0111
1001
-7
6
0110
1010
-6
5
0101
1011
-5
4
0100
1100
-4
3
0011
1101
-3
2
0010
1110
-2
1
0001
1111
-1
0000
0

通过宏来得到具体的取值范围1

```
#include <limits.h>
```

char	int	short	long	long long
SCHAR_MIN SCHAR_MAX	INT_MAX INT_MIN	SHRT_MAX SHRT_MIN	LONG_MAX LONG_MIN	LLONG_MAX LLONG_MIN
UCHAR_MAX	UINT_MAX	USHRT_MAX	ULONG_MAX	ULLONG_MAX
CHAR_MIN CHAR_MAX				

char类型有符号/无符号
由编译器决定

通过宏来得到具体的取值范围2

```
#include <float.h>
```

float	double	long double
FLT_MAX FLT_MIN	DBL_MAX DBL_MIN	LDBL_MAX LDBL_MIN
FLT_DIG	DBL_DIG	LDBL_DIG
FLT_EPSILON	DBL_EPSILON	LDBL_EPSILON
FLT_MAX_EXP FLT_MIN_EXP	DBL_MAX_EXP DBL_MIN_EXP	LDBL_MAX_EXP LDBL_MIN_EXP

实例程序

Demo 4: Overflow example

```
1  printf("%d\n",INT_MAX);
2  printf("%d\n",INT_MAX+1);
3  printf("%d\n",INT_MAX+INT_MAX);
4
5  printf("%d\n",INT_MIN);
6  printf("%d\n",INT_MIN+(-1));
7  printf("%d\n",INT_MIN+INT_MIN);
```

防止溢出

- 如果不需要处理负数，则采用无符号类型。
- 预先估算运算结果的可能范围，采用取值范围更大的类型
- 可能溢出的地方，加入判断语句
- 用ANSI/ISO C定义的宏确定数据的表示范围，解决溢出问题

防止溢出

- 如果不需要处理负数，则采用无符号类型。
- 预先估算运算结果的可能范围，采用取值范围更大的类型
- 可能溢出的地方，加入判断语句
- 用ANSI/ISO C定义的宏确定数据的表示范围，解决溢出问题

防止溢出

- 如果不需要处理负数，则采用无符号类型。
- 预先估算运算结果的可能范围，采用取值范围更大的类型
- 可能溢出的地方，加入判断语句
- 用ANSI/ISO C定义的宏确定数据的表示范围，解决溢出问题

防止溢出

- 如果不需要处理负数，则采用无符号类型。
- 预先估算运算结果的可能范围，采用取值范围更大的类型
- 可能溢出的地方，加入判断语句
- 用ANSI/ISO C定义的宏确定数据的表示范围，解决溢出问题

浮点型的有效位

Demo 5: A simple example of float

```
1  #include <stdio.h>
2  #include <math.h>
3  #define     EPS 1E-5
4  main(){
5       float f;
6       f = 123.456;
7        /*-if(fabs(f-123.456)<EPS)*/
8       if (f == 123.456)
9            printf("f确实等于123.456");
10      else
11           printf("实际上，f等于%f\n", f);
12 }
```

字符型char

- 代表一个数
  - 有符号，代表（-128-127）
  - 无符号，代表（0-255）
- 保存一个字符
  - ASCII（美国标准信息交换码）编码，附录E
  - 可打印字符
  - 不可打印字符（控制字符）

字符型char

- 代表一个数
  - 有符号，代表（-128-127）
  - 无符号，代表（0-255）
- 保存一个字符
  - ASCII（美国标准信息交换码）编码，附录E
  - 可打印字符
  - 不可打印字符（控制字符）

ASC码表

000		(nul)	016	▶	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001	☺	(soh)	017	◀	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002	☻	(stx)	018	↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	♥	(etx)	019	‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	♦	(eot)	020	¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005	♣	(enq)	021	§	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	♠	(ack)	022	▬	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	•	(bel)	023	↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	◘	(bs)	024	↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009		(tab)	025	↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010		(lf)	026		(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	♂	(vt)	027	←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012	♀	(np)	028	∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	¦
013		(cr)	029	↔	(gs)	045	−	061	=	077	M	093	]	109	m	125	}
014	♫	(so)	030	▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	☼	(si)	031	▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂

char型与int型的关系

- char 为一整型数据short short

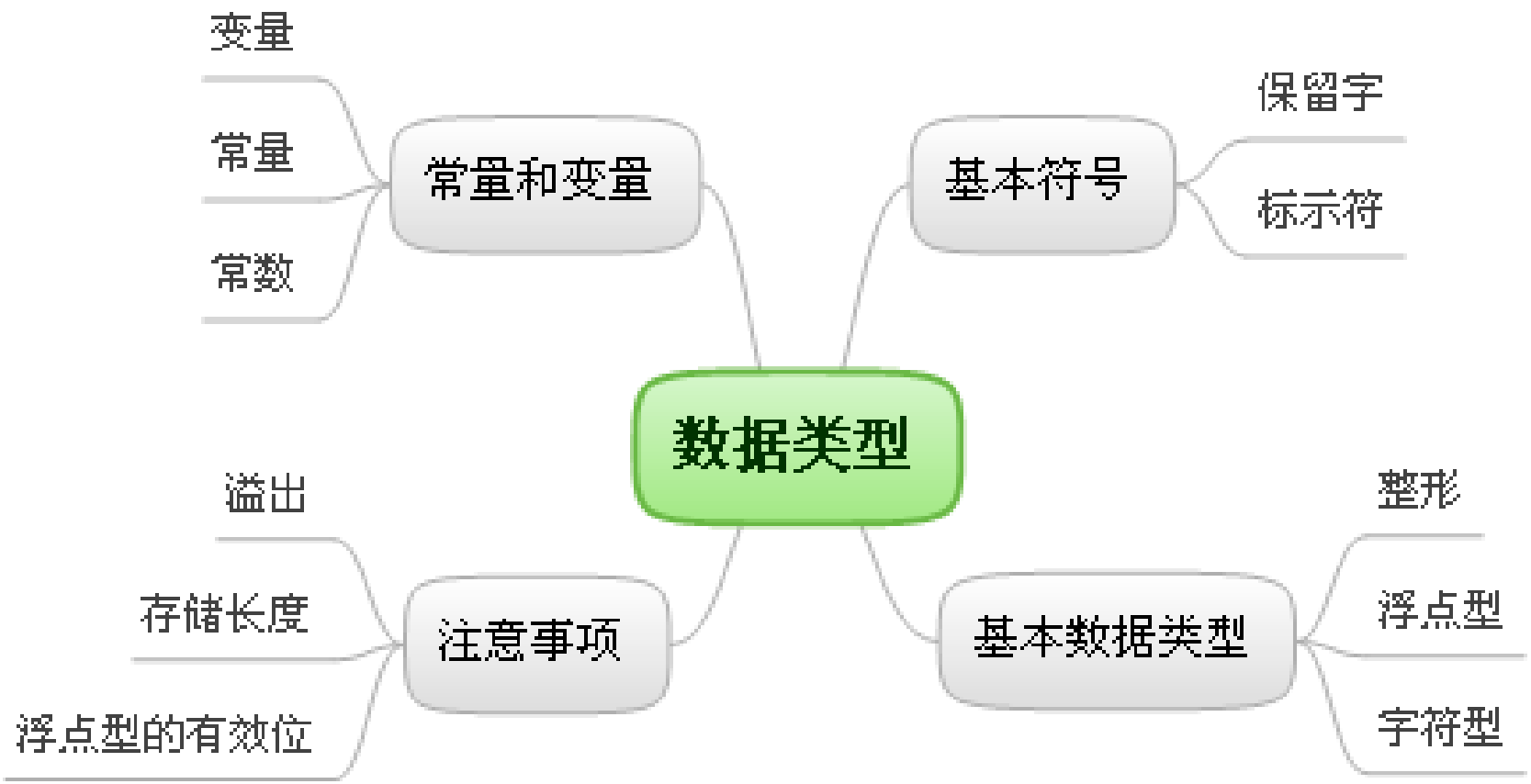
Demo 6: A simple example about char

```
1 char  ch = 'b';
2 printf("%c, %d\n", ch, ch);
3 ch = 'b'-('a'-'A');    /*ch = 'b' - 32;*/
4 printf("%c, %d\n", ch, ch);
```

Demo 7: char or integer

```
1 char c ; /* unsigned char c, what happen? */
2 while((c=getchar())!=EOF){
3 putchar(c);
4 }
```

总结


数据类型
常量和变量
变量
常量
常数
基本符号
保留字
标示符
注意事项
溢出
存储长度
浮点型的有效位
基本数据类型
整形
浮点型
字符型

谢谢大家，欢迎提问！