基础数据类型

huiw@suda.edu.cn

基本数据类型

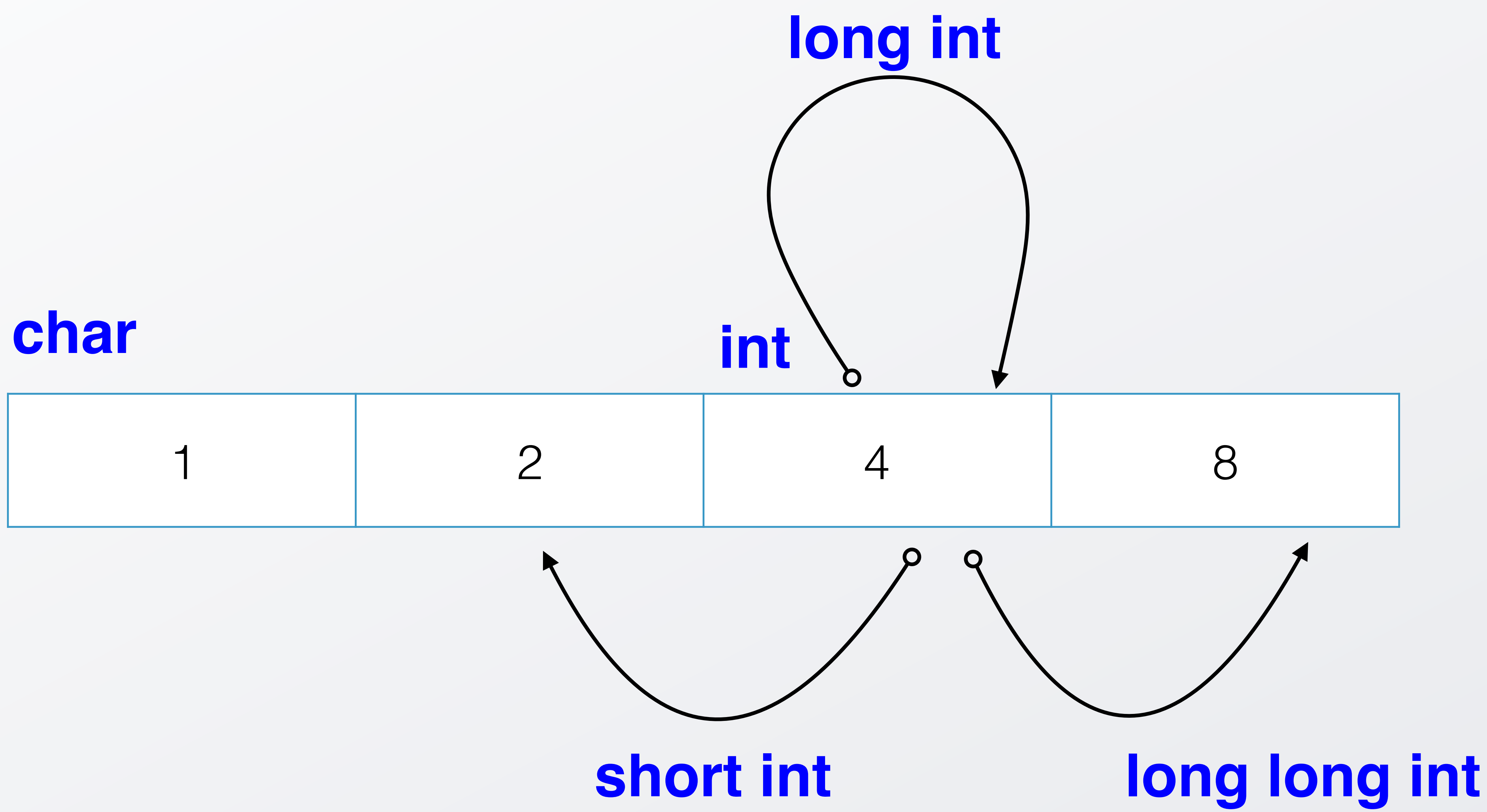
char	字符类型	**1**	可以表示基本（扩展）字符集里面的符号，或0~255的整数*
int	整数类型	**4**	在目前绝大多数机器上
float	单精度浮点数类型	**4**	
double	双精度浮点数类型	**8**	

* 有些系统中使用char类型表示整数时，其隐含取值范围是 -**128 ~ +127**.

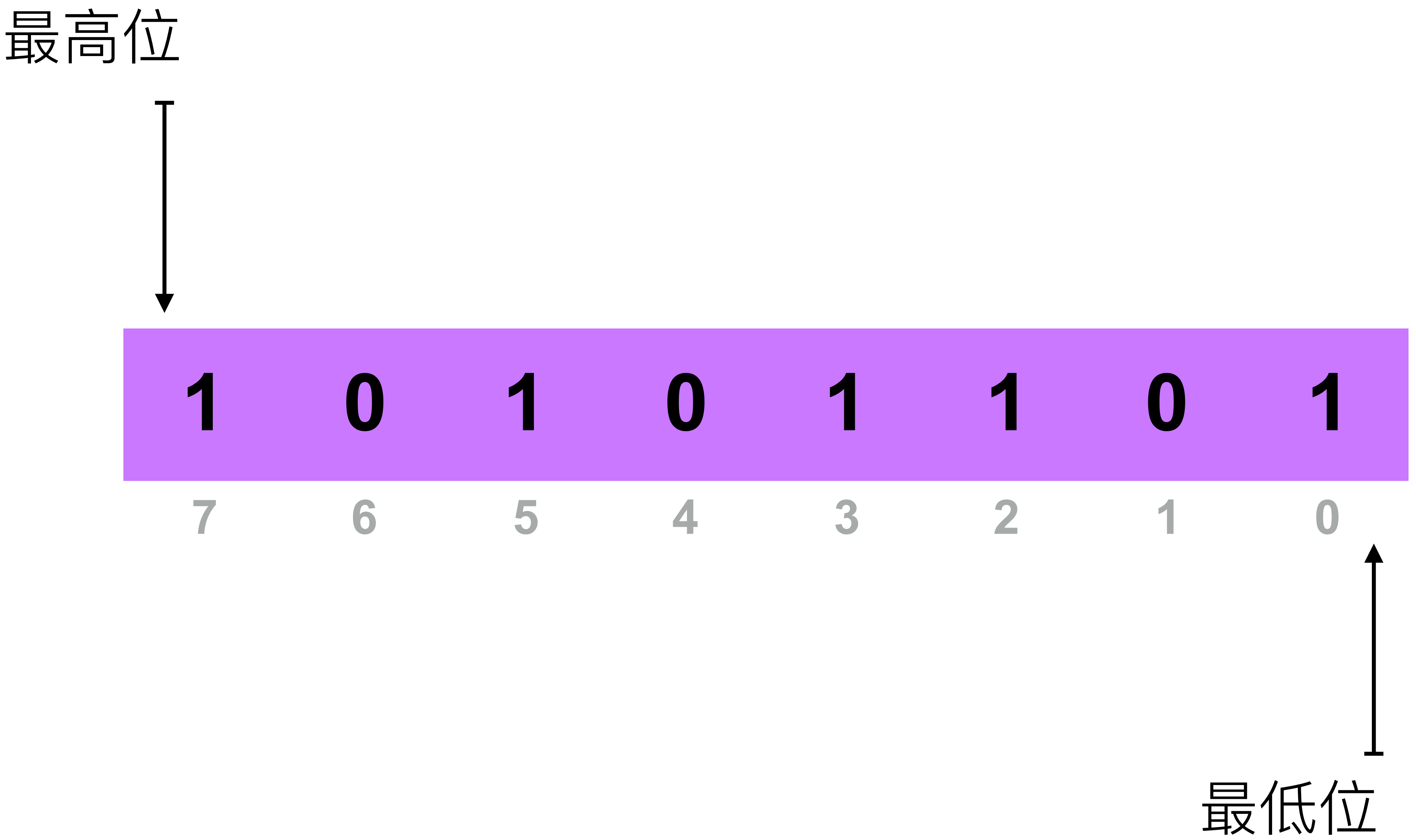
数据类型修饰符

short 和 **long**

数据类型修饰符



先看看一个字节长度的整数char


最高位
1 0 1 0 1 1 0 1
7 6 5 4 3 2 1 0
最低位

1位二进制数可以表示多少个状态?

8位二进制数可以表示多少个状态?

1	**0**	**1**	**0**	**1**	**1**	**0**	**1**	
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
128		**32**		**8**	**4**		**1**	**173**

$$10101101_{(2)} = AD_{(16)} = 173_{(10)}$$

0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

对应的10进制数应该是多少?

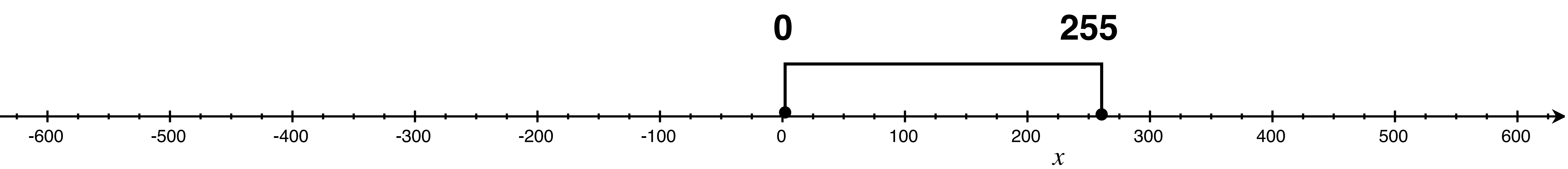
1	1	1	1	1	1	1	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

对应的10进制数应该是多少?

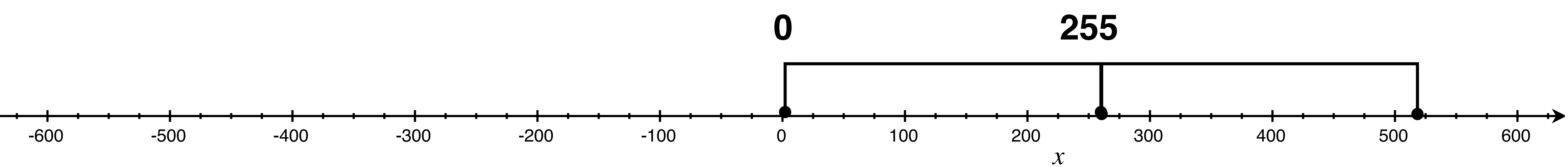
1	**1**	**1**	**1**	**0**	**0**	**0**	**0**
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

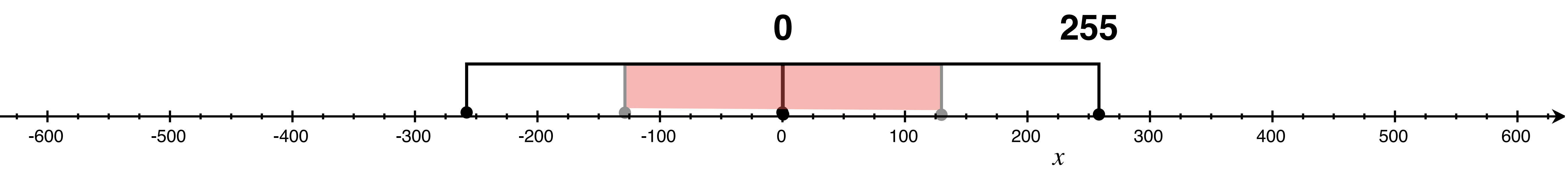
对应的10进制数应该是多少?

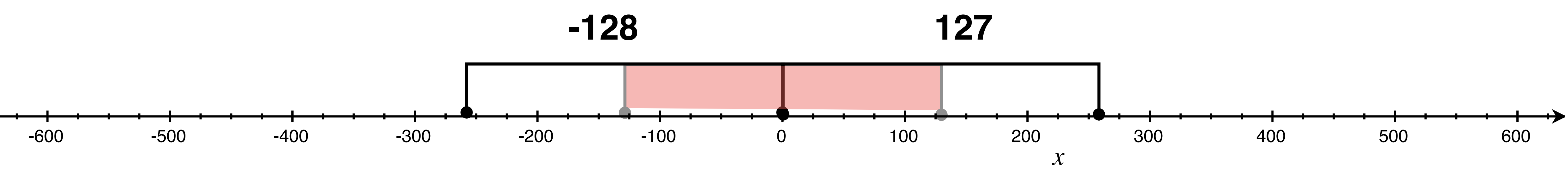
表示负数怎么办?

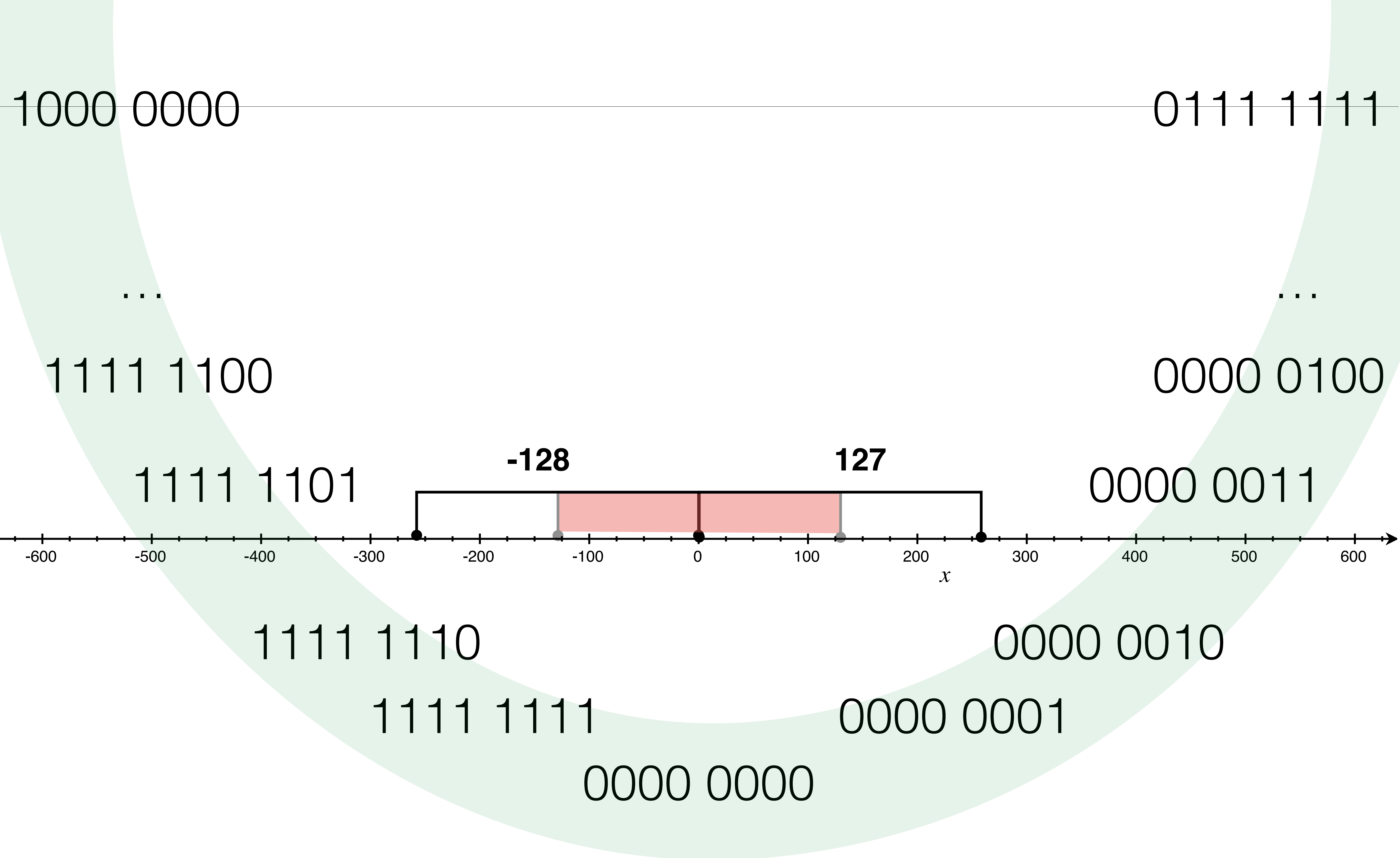


0
255
-600
-500
-400
-300
-200
-100
0
100
200
300
400
500
600
x


0
255
-600
-500
-400
-300
-200
-100
0
100
200
300
400
500
600
x


0
255
-600
-500
-400
-300
-200
-100
0
100
200
300
400
500
600
x


-128
127
-600
-500
-400
-300
-200
-100
0
100
200
300
400
500
600
x


1000 0000
0111 1111
…
…
1111 1100
0000 0100
-128
127
1111 1101
0000 0011
-600
-500
-400
-300
-200
-100
0
100
200
300
400
500
600
x
1111 1110
0000 0010
1111 1111
0000 0001
0000 0000

这种表示方法称为补码表示

补码表示的特点

- 最大为01111111，其真值为 $+127_{(10)}$
- 最小为10000000，其真值为 $-128_{(10)}$

补码表示的特点

- 负数最高位为1
- 正数最高位为0

这样最高位可以看做是 **符号位**。

补码表示的特点

在补码表示法中，0只有一种表示形式：

$[+0]_{补}=00000000$

$[-0]_{补}=11111111+1=00000000$

（由于受设备字长的限制，最后的进位丢失）

所以有$[+0]_{补}=[-0]_{补}=00000000$

数据类型修饰符

signed 和 **unsigned**

数据类型	宽度 (字节)	(位)	最小值	最大值
signed char	**1**	8	-2^7	2^7-1
unsigned char	**1**	8	0	2^8-1
signed short int	**2**	16	-2^{15}	2^7-1
unsigned short int	**2**	16	0	$2^{15}-1$

数据类型	宽度 (字节)	(位)	最小值	最大值
signed int	**4**	32	-2^{31}	$2^{31}-1$
unsigned int	**4**	32	0	$2^{32}-1$
signed long int	**4**	32	-2^{31}	$2^{31}-1$
unsigned long int	**4**	32	0	$2^{32}-1$
signed long long int	**8**	64		
unsigned long long int	**8**	64		

举例

+3与-3的真值

+3

0	**0**	**0**	**0**	**0**	**0**	**1**	**1**
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

按位求反

1	**1**	**1**	**1**	**1**	**1**	**0**	**0**	1

+1

-3

1	**1**	**1**	**1**	**1**	**1**	**0**	**1**	2

-11与+11

-11

1	1	1	1	0	1	0	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

-1

1	1	1	1	0	1	0	0	1

按位求反

+11

0	0	0	0	1	0	1	1	2

编程训练

整数元素的前趋和后继

两个单目运算符

-	~
一元负号	按位求反运算

$$-\ (-3) \rightarrow 3$$

$$\sim 0x01 \rightarrow 0xFE$$

pred_succ.c

```c
int
succ(int n)
{
  return -~n;
}

int
pred(int n)
{
  return ~-n;
}
```

main.c

```c
#include <assert.h>

int pred(int n);
int succ(int n);

int main (int argc, char* argv[]) {

  assert (pred(2) == 1);
  assert (succ(2) == 3);

  assert (pred(-2) == -3);
  assert (succ(-2) == -1);

  assert (pred(0) == -1);
  assert (succ(0) == 1);


  return 0;
}
```

两个源文件名

gcc -g pred_succ.c main.c -o pred_succ

编程训练

加法与减法运算

math_ops.c

```c
#include <assert.h>

int pred(int n);
int succ(int n);


int
add (int a, int b)
{
  assert(b >= 0);
  if (b == 0)
    return a;
  else
    return add(succ(a), pred(b));
}

int
subst(int a, int b)
{
  return add(-b, a);
}
```

main.c

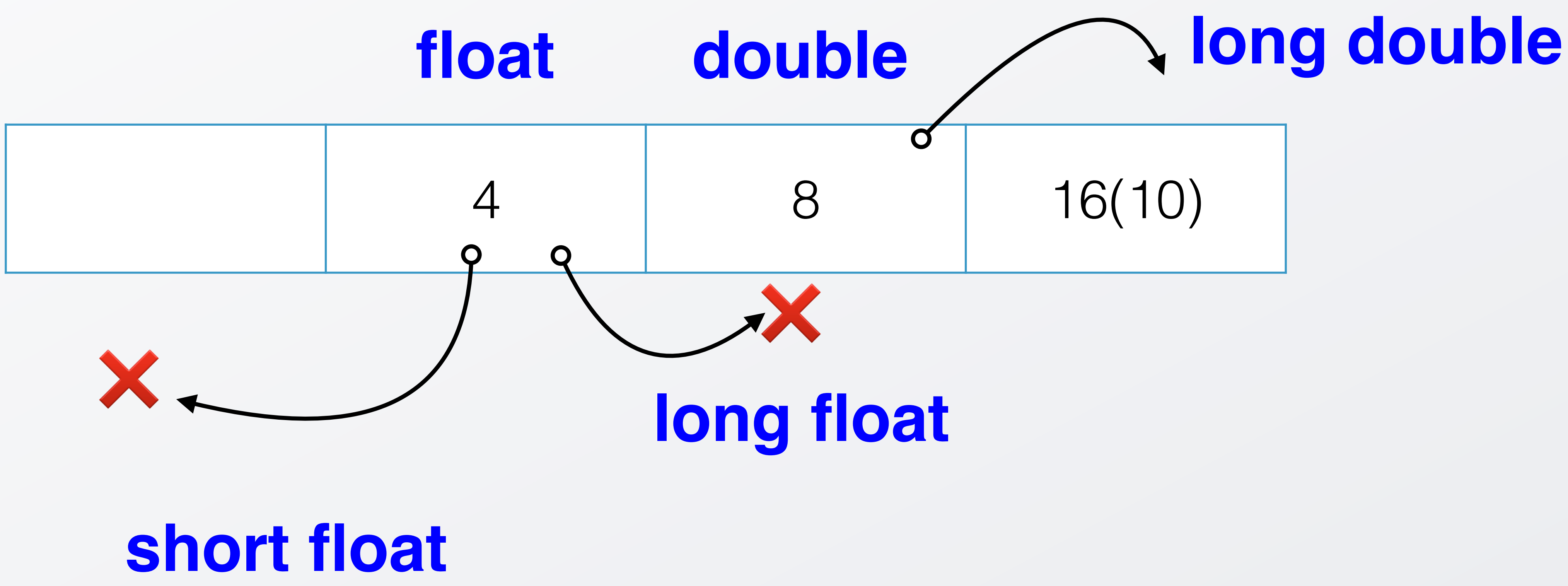
```c
#include <assert.h>

int add(int a, int b);
int subst(int a, int b);

int pred(int n);
int succ(int n);

int main (int argc, char* argv[]) {

  assert (pred(0) == -1);
  assert (succ(0) == 1);
  assert (add(9, 10) == 19);
  assert (subst(17, 10) == 7);


  return 0;
}
```

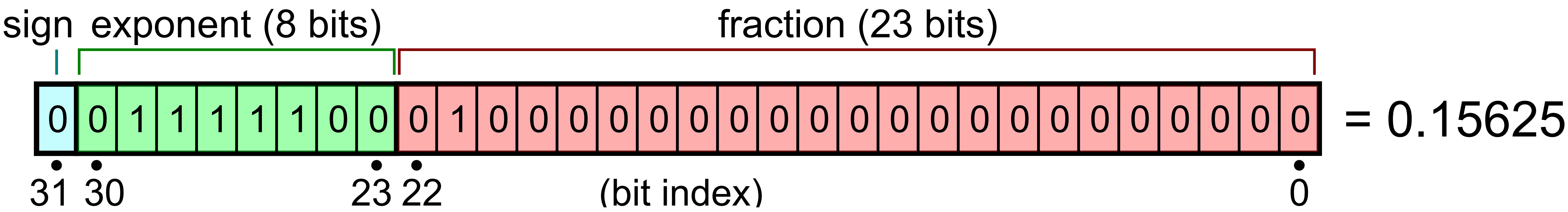
三个源文件名

```
gcc -g pred_succ.c math_ops.c main.c -o pred_succ
```

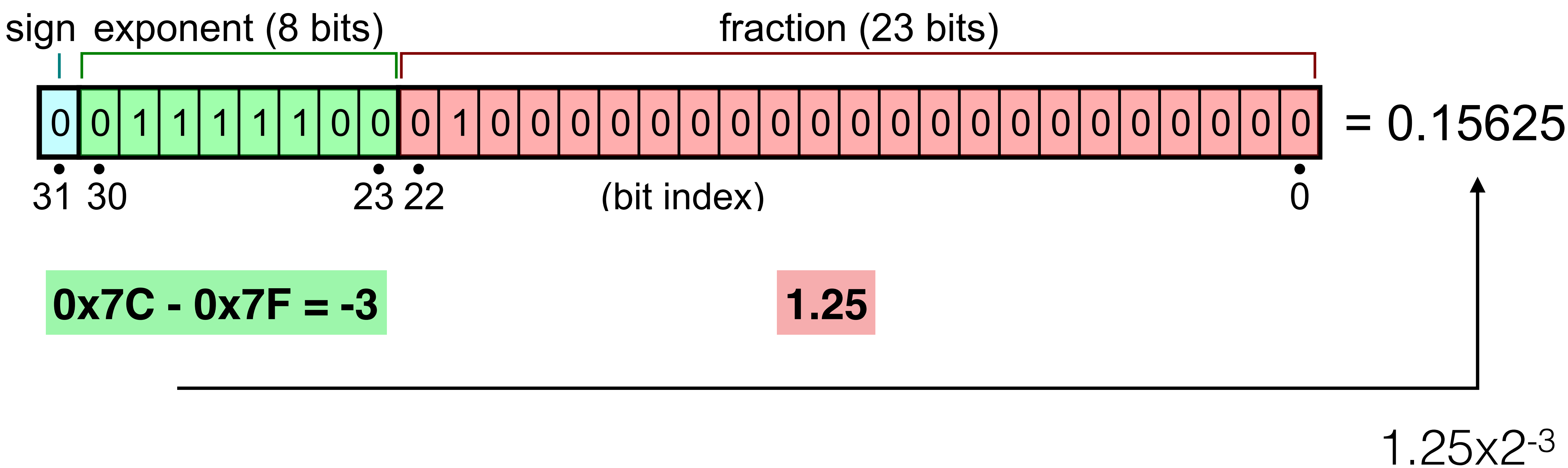
浮点数类型

数据类型修饰符





$$(-1)^{b_{31}} \times (1.b_{22}b_{21}\ldots b_0)_2 \times 2^{(b_{30}b_{29}\ldots b_{23})_2 - 127},$$


sign
exponent (8 bits)
fraction (23 bits)
0 0 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
= 0.15625
31 30 23 22 (bit index) 0
0x7C - 0x7F = -3
1.25
1.25x2-3

浮点数+1.0的内部表示应该是?

$$2^0 \times 1.0$$

0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	**30**							**23**	**22**																						**0**

浮点数-5.0的内部表示应该是?

$$(-1) \times 2^2 \times 1.25$$

1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	**30**							**23**	**22**																						**0**

C语言中的

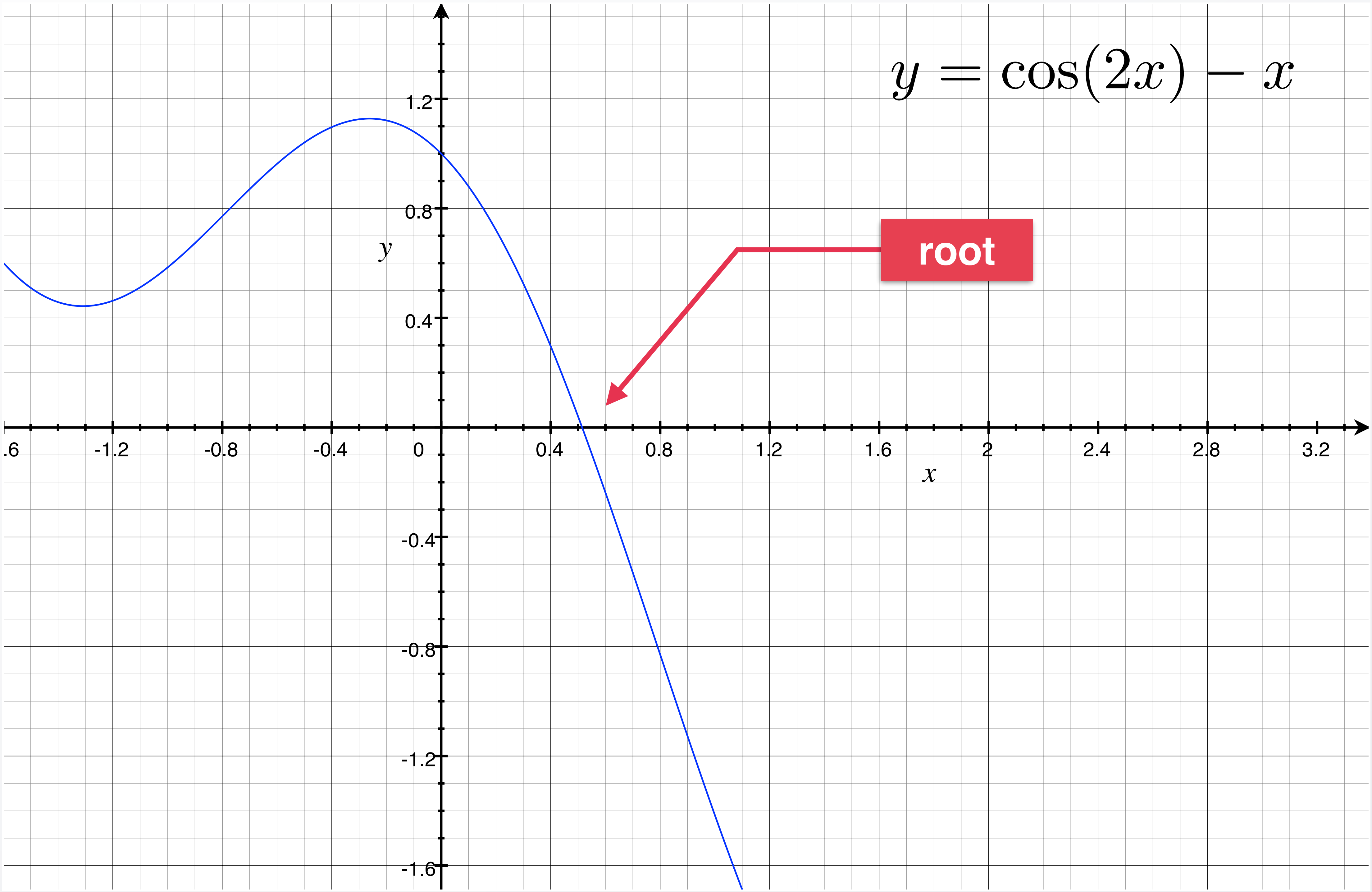
浮点数是离散的，有界的，近似的。

最大的浮点正数

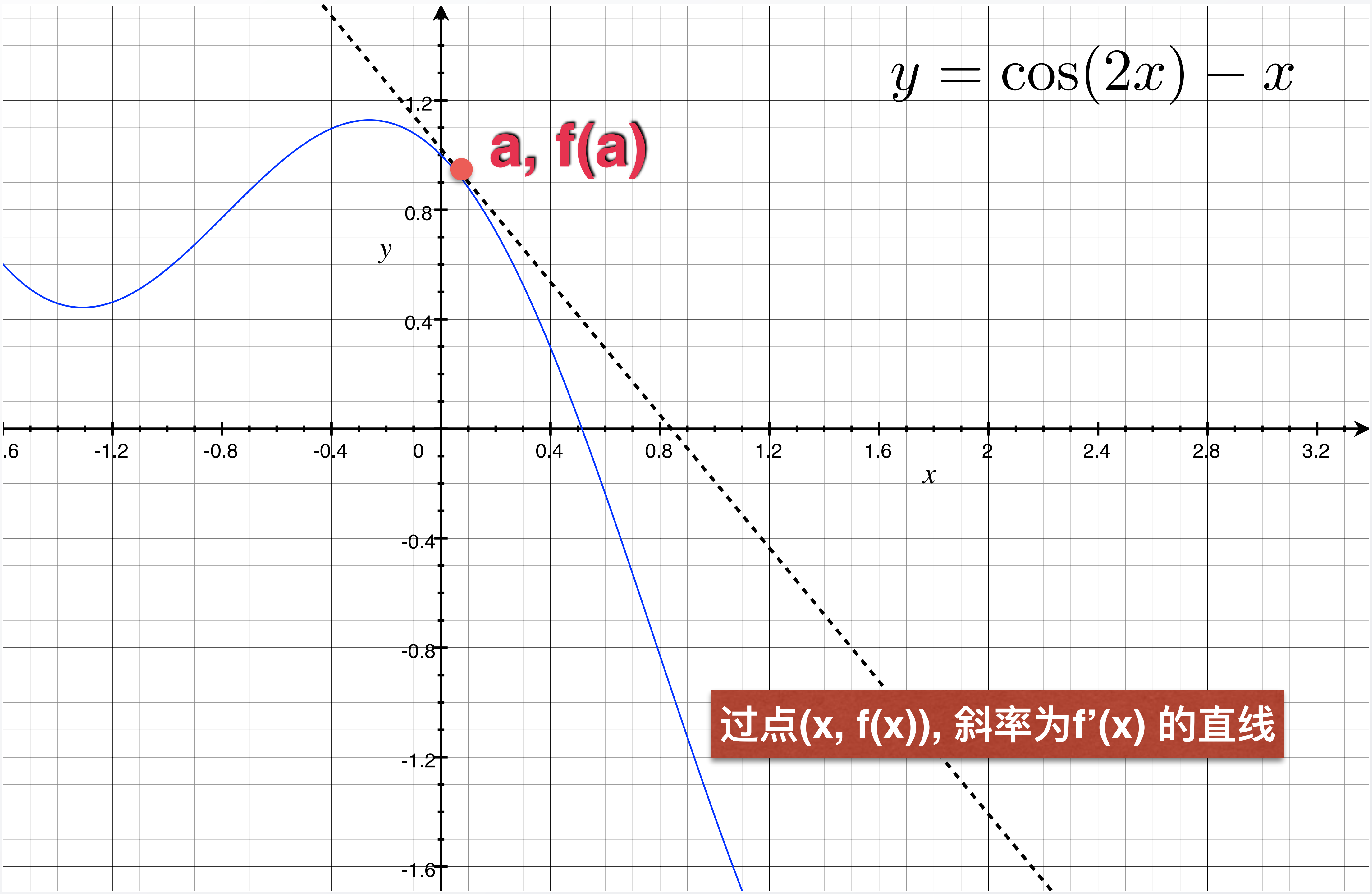
0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
31	**30**							**23**	**22**																						**0**

$$(1 - 2^{-24}) \times 2^{128} \approx 3.402823466 \times 10^{38}$$

程序训练



$y = \cos(2x) - x$
root
1.2
0.8
0.4
-0.4
-0.8
-1.2
-1.6
.6
-1.2
-0.8
-0.4
0
0.4
0.8
1.2
1.6
2
2.4
2.8
3.2
x
y



$y = \cos(2x) - x$
a, f(a)
过点(x, f(x)), 斜率为f’(x) 的直线
x
y
-1.2
-0.8
-0.4
0
0.4
0.8
1.2
1.6
2
2.4
2.8
3.2
1.2
0.8
0.4
-0.4
-0.8
-1.2
-1.6

$$(y - f(a)) = f'(a)(x - a), \quad \text{点斜式}$$

当 $y = 0$,化简上式有

$$\frac{f(a)}{(a - x)} = f'(a)$$

$$f(a) = f'(a)a - f'(a)x$$

$$f'(a)x = f'(a)a - f(a)$$

$$x = a - \frac{f(a)}{f'(a)}$$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

newton.c

```c
#include <stdio.h>
#include <stdbool.h>
#include <math.h>

double f(double x) {
       return ( cos(2*x) - x );
}

double f_prime(double x).
{
   return ( -2*sin(2*x)-1 );.
}
```

newton.c

```c
bool is_good_enough(double x) {

  const double epilon = 1e-4;
  if (fabs(f(x)) < epilon)
    return true;

  return false;

}
```

newton.c

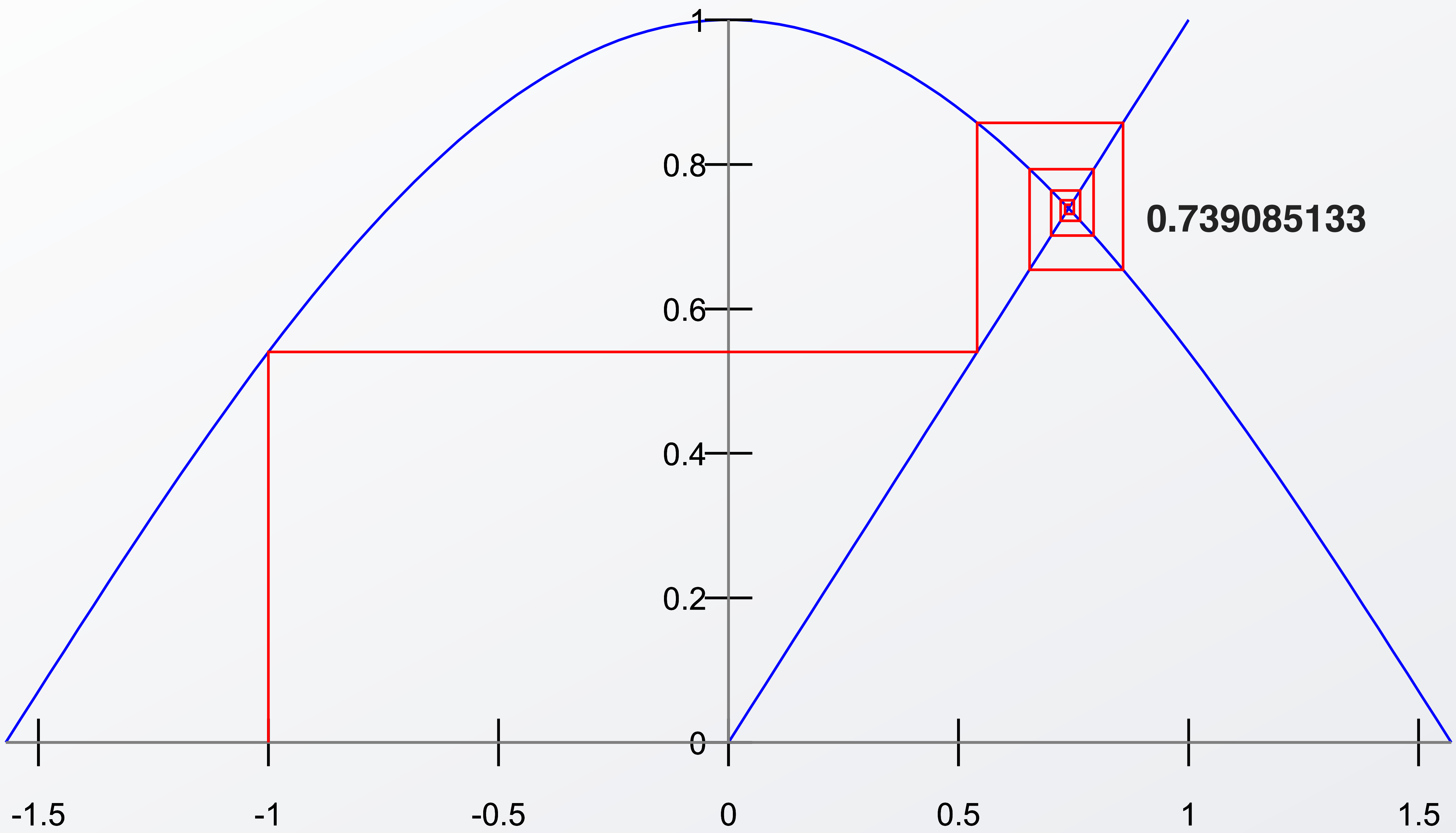
```c
double iterate(double x_prev) {

  double new_x = x_prev - f(x_prev) / f_prime(x_prev);

  if (is_good_enough(new_x))
    return new_x;

  return iterate(new_x);
}
```

main.c

```c
#include <stdio.h>

double iterate(double x_prev);

int main (int argc, char* argv[]) {

  printf("%f\n", iterate(1.0));

  return 0;
}
```

计算函数的不动点(fixed-point)

$$cos(\ \ldots\ cos(cos(cos(-1.0))))$$



1
0.8
0.6
0.4
0.2
0
0.739085133
-1.5
-1
-0.5
0
0.5
1
1.5

如果

$$x = f(x)$$

则称 x 为函数 f 的一个不动点。

fixed_point.c

```c
#include <stdio.h>
#include <math.h>
#include <stdbool.h>

const double tolerance = 0.00001f;

typedef double (*fp_t) (double x);

bool
is_close_enough(double x1, double x2) {
  if (fabs(x1 - x2) < tolerance)
    return true;

  return false;
}

double try(fp_t f, double guess) {
  double new_guess = f(guess);
  if (is_close_enough(new_guess, guess))
    return new_guess;

  return try(f, new_guess);
}
```

fixed_point.c

```c
double fixed_point(fp_t f, double first_guess ){
  return try(f, first_guess);
}
```

main.c

```c
#include <stdio.h>
#include <math.h>
typedef double (*fp_t) (double x);

double fixed_point(fp_t f, double first_guess );

double golden_ratio(double x) {
  return 1.0 + 1.0/x;
}

/*
 * x^x = 1000
 * x = log(1000)/log(x).
 *.
 */
double gingle_bell(double x){
  return log(1000) / log(x);
}


int main (int argc, char* argv[]) {
  printf("%lf\n", fixed_point(golden_ratio, 1.0f));
  printf("%lf\n", fixed_point(gingle_bell, 2.0f));
  return 0;
}
```

字符类型

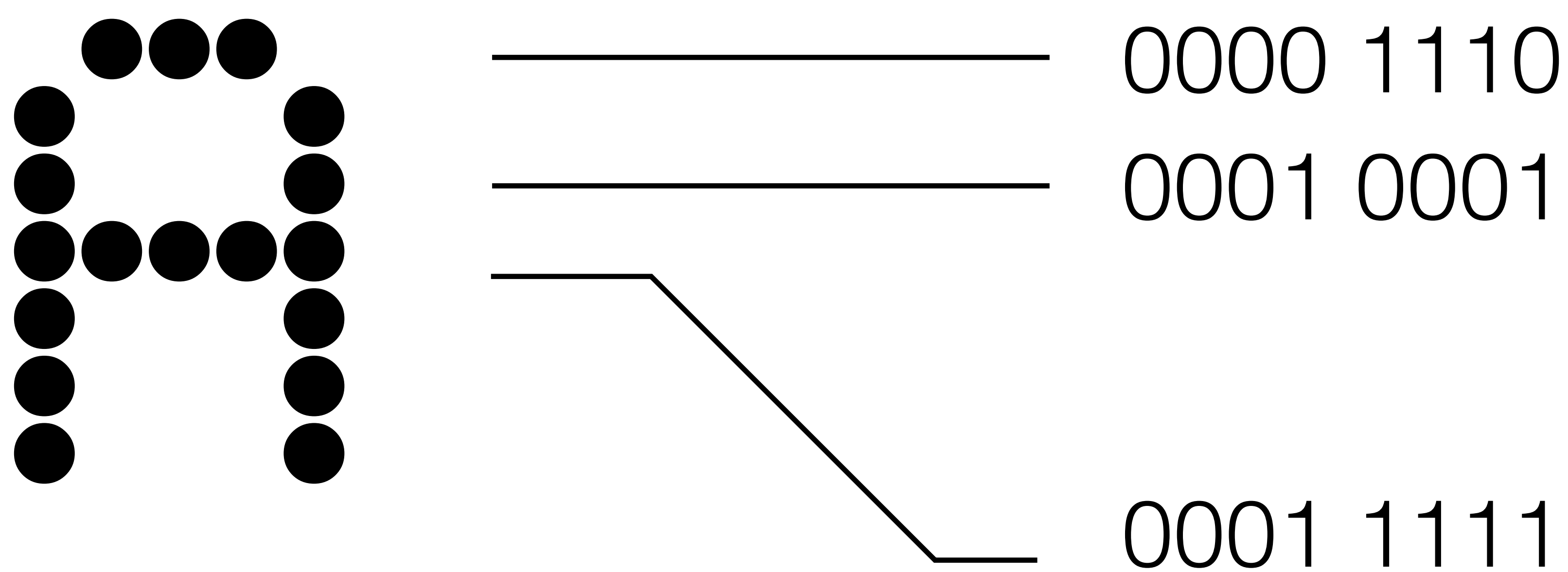
	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	NUL 0000 *0*	SOH 0001 *1*	STX 0002 *2*	ETX 0003 *3*	EOT 0004 *4*	ENQ 0005 *5*	ACK 0006 *6*	BEL 0007 *7*	BS 0008 *8*	HT 0009 *9*	LF 000A *10*	VT 000B *11*	FF 000C *12*	CR 000D *13*	SO 000E *14*	SI 000F *15*
1_	DLE 0010 *16*	DC1 0011 *17*	DC2 0012 *18*	DC3 0013 *19*	DC4 0014 *20*	NAK 0015 *21*	SYN 0016 *22*	ETB 0017 *23*	CAN 0018 *24*	EM 0019 *25*	SUB 001A *26*	ESC 001B *27*	FS 001C *28*	GS 001D *29*	RS 001E *30*	US 001F *31*
2_	SP 0020 *32*	! 0021 *33*	" 0022 *34*	# 0023 *35*	$ 0024 *36*	% 0025 *37*	& 0026 *38*	' 0027 *39*	(0028 *40*	) 0029 *41*	* 002A *42*	+ 002B *43*	, 002C *44*	- 002D *45*	. 002E *46*	/ 002F *47*
3_	0 0030 *48*	1 0031 *49*	2 0032 *50*	3 0033 *51*	4 0034 *52*	5 0035 *53*	6 0036 *54*	7 0037 *55*	8 0038 *56*	9 0039 *57*	: 003A *58*	; 003B *59*	< 003C *60*	= 003D *61*	> 003E *62*	? 003F *63*
4_	@ 0040 *64*	A 0041 *65*	B 0042 *66*	C 0043 *67*	D 0044 *68*	E 0045 *69*	F 0046 *70*	G 0047 *71*	H 0048 *72*	I 0049 *73*	J 004A *74*	K 004B *75*	L 004C *76*	M 004D *77*	N 004E *78*	O 004F *79*
5_	P 0050 *80*	Q 0051 *81*	R 0052 *82*	S 0053 *83*	T 0054 *84*	U 0055 *85*	V 0056 *86*	W 0057 *87*	X 0058 *88*	Y 0059 *89*	Z 005A *90*	[005B *91*	\ 005C *92*	] 005D *93*	^ 005E *94*	_ 005F *95*
6_	` 0060 *96*	a 0061 *97*	b 0062 *98*	c 0063 *99*	d 0064 *100*	e 0065 *101*	f 0066 *102*	g 0067 *103*	h 0068 *104*	i 0069 *105*	j 006A *106*	k 006B *107*	l 006C *108*	m 006D *109*	n 006E *110*	o 006F *111*
7_	p 0070 *112*	q 0071 *113*	r 0072 *114*	s 0073 *115*	t 0074 *116*	u 0075 *117*	v 0076 *118*	w 0077 *119*	x 0078 *120*	y 0079 *121*	z 007A *122*	{ 007B *123*	| 007C *124*	} 007D *125*	~ 007E *126*	DEL 007F *127*

```c
char letter_h = 'H';
char letter_k = 'K';
```

```
char char_0 = '0';
char char_4 = ‘4';
char left_square_bracket = ‘\x5B’;
char new_line_ctrl_char = ‘\n';
```

Escape sequence	Hex value in ASCII	Character represented
`\a`	7	Alert (Beep, Bell) (added in C89)[1]
`\b`	8	Backspace
`\f`	0C	Formfeed
`\n`	0A	Newline (Line Feed); see notes below
`\r`	0D	Carriage Return
`\t`	9	Horizontal Tab
`\v`	0B	Vertical Tab
`\\`	5C	Backslash
`\'`	27	Single quotation mark
`\"`	22	Double quotation mark
`\?`	3F	Question mark (used to avoid trigraphs)
`\nnn`note 1	any	The byte whose numerical value is given by *nnn* interpreted as an octal number
`\xhh…`	any	The byte whose numerical value is given by *hh…* interpreted as a hexadecimal number
`\e`note 2	1B	escape character (some character sets)
`\Uhhhhhhhh`note 3	none	Unicode code point where *h* is a hexadecimal digit
`\uhhhh`note 4	none	Unicode code point below 10000 hexadecimal

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789


0000 1110
0001 0001
0001 1111

dot_mat.c

```
#include <stdio.h>

int
print_dot_line(char mat_code, unsigned int counter)
{
  if (counter > 5) return putchar('\n');

  if (mat_code & 1)
    putchar('*');
  else
    putchar(' ');

  return print_dot_line(mat_code >> 1, counter + 1);

}
```

dot_mat.c

```c
char
prt_letter_a()
{
  print_dot_line('\x0E', 1);
  print_dot_line('\x11', 1);
  print_dot_line('\x11', 1);
  print_dot_line('\x1F', 1);
  print_dot_line('\x11', 1);
  print_dot_line('\x11', 1);
  print_dot_line('\x11', 1);
  return 'A';
}
```

main.c

```c
char
prt_letter_a();

int main (int argc, char* argv[]) {
  prt_letter_a();
  return 0;
}
```

编程计算$sin(x)$

$$sin(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}$$

$$= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots$$

抽象数据类型

函数 与 复合数据类型

func()

unsigned double
char signed
float long short

存储操作

int
= unsigned int

& ~ -(单目)

运算符与表达式

+ - * / % ! || && == != < > <= >=

程序流程控制

goto if…else…


cos
fabs
putchar
库函数

WENZHENG COLLEGE OF SOOCHOW UNIVERSITY

2017.3.29

Soochow University

附录

Name	From	Description
`<assert.h>`		Contains the assert macro, used to assist with detecting logical errors and other types of bug in debugging versions of a program.
`<complex.h>`	C99	A set of functions for manipulating complex numbers.
`<ctype.h>`		Defines set of functions used to classify characters by their types or to convert between upper and lower case in a way that is independent of the used
`<errno.h>`		For testing error codes reported by library functions.
`<fenv.h>`	C99	Defines a set of functions for controlling floating-point environment.
`<float.h>`		Defines macro constants specifying the implementation-specific properties of the floating-point library.
`<inttypes.h>`	C99	Defines exact width integer types.
`<iso646.h>`	NA1	Defines several macros that implement alternative ways to express several standard tokens. For programming in ISO 646 variant character sets.
`<limits.h>`		Defines macro constants specifying the implementation-specific properties of the integer types.
`<locale.h>`		Defines localization functions.
`<math.h>`		Defines common mathematical functions.
`<setjmp.h>`		Declares the macros `setjmp` and `longjmp`, which are used for non-local exits.
`<signal.h>`		Defines signal handling functions.
`<stdalign.h>`	C11	For querying and specifying the alignment of objects.
`<stdarg.h>`		For accessing a varying number of arguments passed to functions.
`<stdatomic.h>`	C11	For atomic operations on data shared between threads.
`<stdbool.h>`	C99	Defines a boolean data type.
`<stddef.h>`		Defines several useful types and macros.
`<stdint.h>`	C99	Defines exact width integer types.
`<stdio.h>`		Defines core input and output functions

<locale.h>		Defines localization functions.
<math.h>		Defines common mathematical functions.
<setjmp.h>		Declares the macros `setjmp` and `longjmp`, which are used for non-local exits.
<signal.h>		Defines signal handling functions.
<stdalign.h>	C11	For querying and specifying the alignment of objects.
<stdarg.h>		For accessing a varying number of arguments passed to functions.
<stdatomic.h>	C11	For atomic operations on data shared between threads.
<stdbool.h>	C99	Defines a boolean data type.
<stddef.h>		Defines several useful types and macros.
<stdint.h>	C99	Defines exact width integer types.
<stdio.h>		Defines core input and output functions
<stdlib.h>		Defines numeric conversion functions, pseudo-random numbers generation functions, memory allocation, process control functions
<stdnoreturn.h>	C11	For specifying non-returning functions.
<string.h>		Defines string handling functions.
<tgmath.h>	C99	Defines type-generic mathematical functions.
<threads.h>	C11	Defines functions for managing multiple Threads as well as mutexes and condition variables.
<time.h>		Defines date and time handling functions
<uchar.h>	C11	Types and functions for manipulating Unicode characters.
<wchar.h>	NA1	Defines wide string handling functions.
<wctype.h>	NA1	Defines set of functions used to classify wide characters by their types or to convert between upper and lower case

Type	Explanation	Format Specifier
char	Smallest addressable unit of the machine that can contain basic character set. It is an integer type. Actual type can be either signed or unsigned. It contains `CHAR_BIT` bits.[3]	%c
signed char	Of the same size as `char`, but guaranteed to be signed. Capable of containing **at least** the [−127, +127] range;[3][4]	%c (or %hhi for numerical output)
unsigned char	Of the same size as `char`, but guaranteed to be unsigned. It is represented in binary notation without padding bits; thus, its range is exactly [0, $2^{\texttt{CHAR\_BIT}}$ − 1].[5]	%c (or %hhu for numerical output)
short short int signed short signed short int	*Short* signed integer type. Capable of containing **at least** the [−32,767, +32,767] range;[3][4] thus, it is at least 16 bits in size. The negative value is −32767 (not −32768) due to the one's-complement and sign-magnitude representations allowed by the standard, though the two's-complement representation is much more common.[6]	%hi
unsigned short unsigned short int	*Short* unsigned integer type. Contains **at least** the [0, 65535] range;[3][4]	%hu
int signed signed int	Basic signed integer type. Capable of containing **at least** the [−32,767, +32,767] range;[3][4] thus, it is at least 16 bits in size.	%i or %d
unsigned unsigned int	Basic unsigned integer type. Contains **at least** the [0, 65535] range;[3][4]	%u
long long int signed long signed long int	*Long* signed integer type. Capable of containing **at least** the [−2,147,483,647, +2,147,483,647] range;[3][4] thus, it is at least 32 bits in size.	%li
unsigned long unsigned long int	*Long* unsigned integer type. Capable of containing **at least** the [0, 4,294,967,295] range;[3][4]	%lu
long long long long int signed long long signed long long int	*Long long* signed integer type. Capable of containing **at least** the [−9,223,372,036,854,775,807, +9,223,372,036,854,775,807] range;[3][4] thus, it is at least 64 bits in size. Specified since the C99 version of the standard.	%lli
unsigned long long unsigned long long int	*Long long* unsigned integer type. Contains **at least** the [0, +18,446,744,073,709,551,615] range;[3][4] Specified since the C99 version of the standard.	%llu
float	Real floating-point type, usually referred to as a single-precision floating-point type. Actual properties unspecified (except minimum limits), however on most systems this is the IEEE 754 single-precision binary floating-point format. This format is required by the optional Annex F "IEC 60559 floating-point arithmetic".	%f (promoted automatically to `double` for `printf()`)
double	Real floating-point type, usually referred to as a double-precision floating-point type. Actual properties unspecified (except minimum limits), however on most systems this is the IEEE 754 double-precision binary floating-point format. This format is required by the optional Annex F "IEC 60559 floating-point arithmetic".	%f (%F) (%lf (%lF) for `scanf()`) %g %G %e %E (for scientific
long double	Real floating-point type, usually mapped to an extended precision floating-point number format. Actual properties unspecified. Unlike types float and double, it can be either 80-bit floating point format, the non-IEEE "double-double" or IEEE 754 quadruple-precision floating-point format if a higher precision format is provided, otherwise it is the same as double. See the article on long double for details.	%Lf %LF %Lg %LG %Le %LE[7]