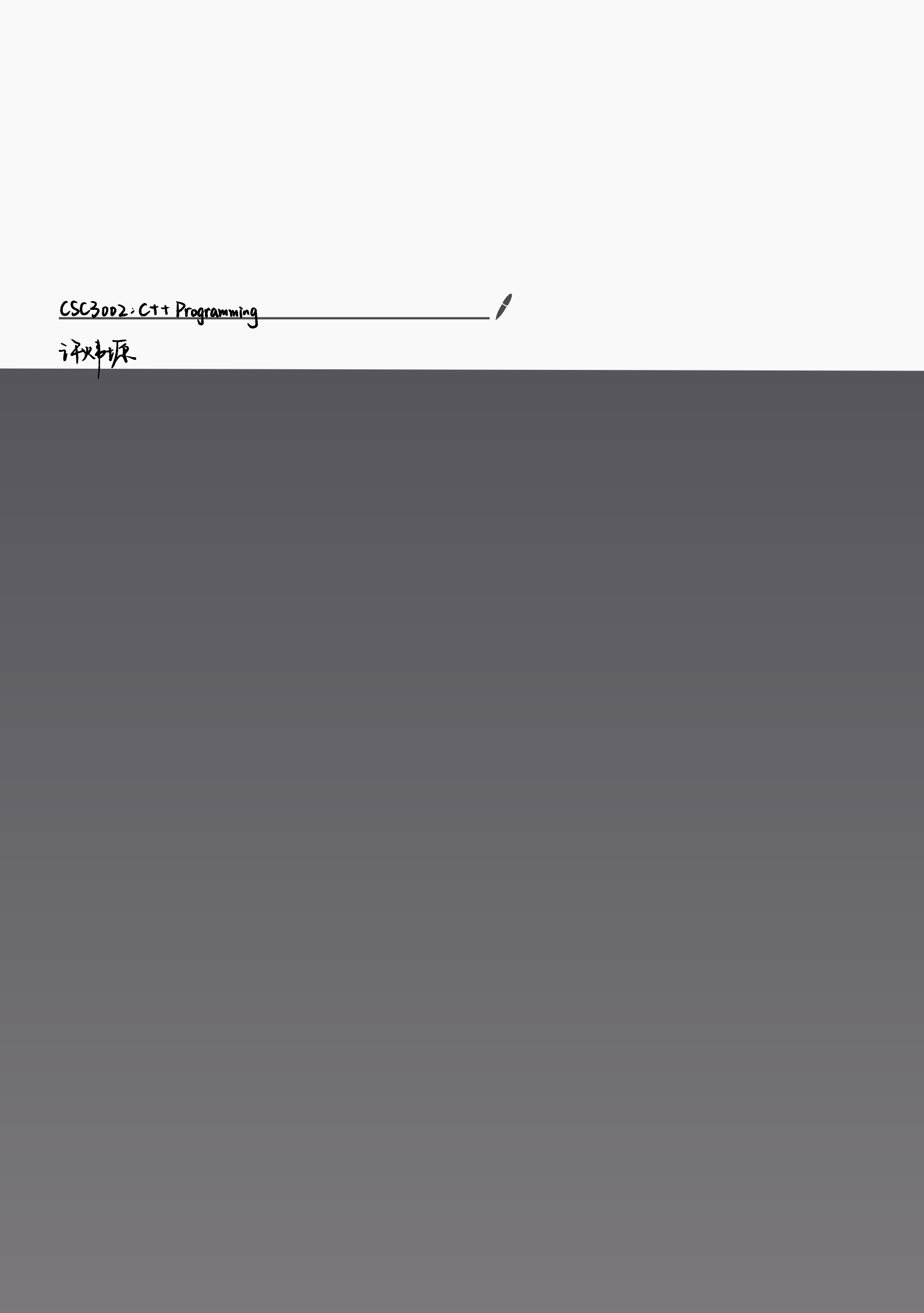
CSC3002: C++ Programming



L1-L3 Introduction to C++

1. C++ Context

① C++ & py

	C++	Python
Execution	Compiled and linked into an executable.	Interpreted and executed by an engine.
Memory Management	Requires much more attention to bookkeeping and storage details	Nearly no attention to be paid to memory management. Supports garbage collection.
Types	Static typing, explicitly declared, bound to names, checked at compile time	Dynamic typing, bound to values, checked at run time, and are not so easily subverted
Language Complexity	Complex and full-featured. C++ tries to give you every language feature under the sun while at the same time never (forcibly) abstracting anything away that could potentially affect performance.	Simple. Python tries to give you only one or a few ways to do things, and those ways are designed to be simple, even at the cost of some language power or running efficiency.

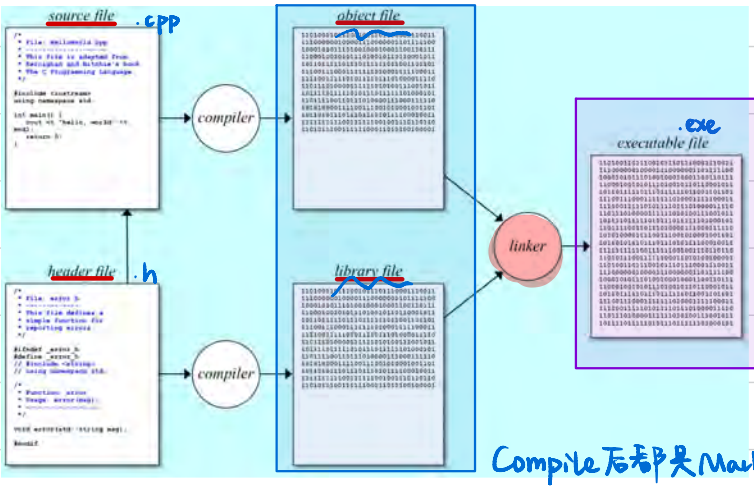
Java

Compiler + Interpreter

② Compiler & Interpreter

a. Complier

Program ⟶ Object Code (Binary Code) ⟶ Implementation



最后只需要这个

Compile后都是Machine Language

b. Interpreter

Program ⟶ Implementation

Complier：一次性的翻译（编译器）⟶更快

Interpreter：逐行翻译（口译）

③ IDE

Integrated Development Environment ≙ Software (vsc)

≙
- Code editor (Including GUI)
- Compiler + Interpreter + Linker
- Debugger

2. C++ Components

① Example

```
/*
 * File: PowersOfTwo.cpp
 * ---------------------
 * This program generates a list of the powers of
 * two up to an exponent limit entered by the user.
 */

#include <iostream>
using namespace std;

/* Function prototypes */

int raiseToPower(int n, int k);

/* Main program */

int main() {
   int limit;
   cout << "This program lists powers of two." << endl;
   cout << "Enter exponent limit: ";
   cin >> limit;
   for (int i = 0; i <= limit; i++) {
      cout << "2 to the " << i << " = "
           << raiseToPower(2, i) << endl;
   }
   return 0;
}
```

—— Comments

—— library

—— declaration of functions

—— main function

② Comments

- Multi-line comments

```
/* Comments line 1
   Comments line 2
   Comments line 3 */
```

- Single-line comments

```
// Comments line 1
// Comments line 2
// Comments line 3
```

③ Library

< > 内一定是 Std Library

- `#include <iostream>` instructs the compiler to read the relevant definitions from what is called a header (.h) file; for you own libraries, write `#include "myownlib.h"`. (Remember `import` in Python?)
- To ensure that the names defined in different parts of a large system do not interfere with one another, C++ segments code into structures called namespaces, each of which keeps track of its own set of names.
- The standard C++ libraries use a namespace called `std`, which means that you cannot refer to the names defined in standard header files like *iostream.h* unless you let the compiler know to which namespace those definitions belong.

用于区分不同的同名函数

Using "`using namespace`"

- To avoid always using the entire namespace, one can also use specific using declarations for just the names you need, e.g., "`using x::name1; using x::name2;`"
- You can declare and put things into your own namespace, but in this course we mostly just need "`using namespace std`", or "`using std::cout; using std::endl;`".

④ Functions

Every C++ program must contain a function with the name `main`. It specifies the starting point for the computation and is called when the program starts up. When `main` has finished its work and returns, execution of the program ends.

functions 可以直接 define (declare + assign)

不是一直都可以实现，如 mutual recursion

先 D 后 A

The "declare-before-use" rule

- A C++ program consists of various entities such as variables, functions, types, and namespaces. Each of these entities must be declared before they can be used. (Think about the inclusion of libraries in the very beginning of the program).
- Notice that function `raiseToPower` is defined in the function definition part after the `main` function. To comply with the "declare-before-use" rule, a declaration of the function, called a function prototype, must appear before the `main` function.
- The function prototype specifies a unique name for the function, along with information about its type and other characteristics (more details later).

如此 ①

② 大项目下代码更易读

3. Data Types

① Category

- Primitive Type
- Reference Type

② Typical Primitive Type

`int`	This type is used to represent integers, which are whole numbers such as 17 or −53.
`double`	This type is used to represent numbers that include a decimal fraction, such as 3.14159265.
`bool`	This type represents a logical value (`true` or `false`).
`char`	This type represents a single ASCII character.

③ Named Constants

- To create a named constant in C++, precede the regular variable declaration with the keyword `const`. This tells the compiler that a special variable has been created which has a value that is invariable. 在 Datatype 前

```
const double PI = 3.14159265358979323846;
```

- When creating a named constant, it must be assigned a value.

variable value 不改
variable 不受重定义影响

4. Operators

Binary
Unary

① Basic operators

+	Addition	*	Multiplication
−	Subtraction	/	Division
		%	Remainder

若是整数相除，则向下取整

```
double c = 38;
double f = 9 / 5 * c + 32;
```
→ ✗

```
double c = 38;
double f = double(9) / 5 * c + 32;
```
→ ✓

```
double(14)/5
```
↓ 2.8

```
double(14/5)
```
↓ 2.0

② Logic operator

- && || → 先判断左边，再判断右边 (更快)
- & | → 同时判断

③ ? : Operator

condition ? *expression*$_1$: *expression*$_2$

false → *expression*$_2$
true → *expression*$_1$

```
max = (x > y) ? x : y;
```

④ Switch Operator

```
switch ( expression ) {
  case v1:
    statements to be executed if expression = v1
    break;
  case v2:
    statements to be executed if expression = v2
    break;
  . . . more case clauses if needed . . .
  default:
    statements to be executed if no values match
    break;
}
```

= else

Sequential

L4 Functions & Libraries

1. Function

① Overloading & Signature

Overload: Same function name while different signature

Signature: The number and types of arguments (Not the name)

② Default value

a. Requirements

~~Default values appear only in the function prototype, not in the function definition~~ 可以在implement的时候设置Default values

Optional parameters (有argument with default value) must appear at the end

↳ 这样的conversion了是为了compiler可以顺序运行

b. Example

Two codes are not the same: 1st: Allows 0111 inputs; 2nd: Allows 012 inputs

⚠ 1st code

```
void setInitialLocation(double x = 0, double y = 0);
```

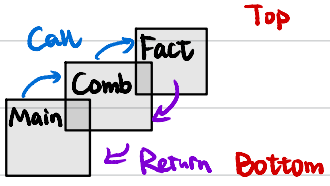
⚠ 2nd code (Overloading)

```
void setInitialLocation(double x, double y);

void setInitialLocation() {
   setInitialLocation(0, 0);
}
```

③ The essence of calling functions

Stack frame



Call = Put

Return = Pop

④Reference variables (由swap()引出)

a. Definition

As we knew before, there are { primitive variable
reference variable (reference variable 与某个variable dependent)

mutually affect

```
int n1 = 1, n2 = 2;
int & x = n1, & y = n2; // x = 1, y = 2
n1 = 2, n2 = 1; // x = 2, y = 1
x = 1, y = 2; // n1 = 1, n2 = 2
```

b. Requirements

{ A reference variable must be initialized when it is declared
Once a reference variable is initialized with a variable, it cannot be reassigned to another variable

c. Example

```
int n1 = 1, n2 = 2;
swap(n1, n2);
```

(X)

```
void swap(int x, int y) {
    int tmp = x;
    x = y;
    y = tmp;
}
```

(没有return出来，改的只是local variables)

(√)

```
void swap(int & x, int & y) {
    int tmp = x;
    x = y;
    y = tmp;
}
```

(获得了reference variable，建立了local & global variables间的联系)

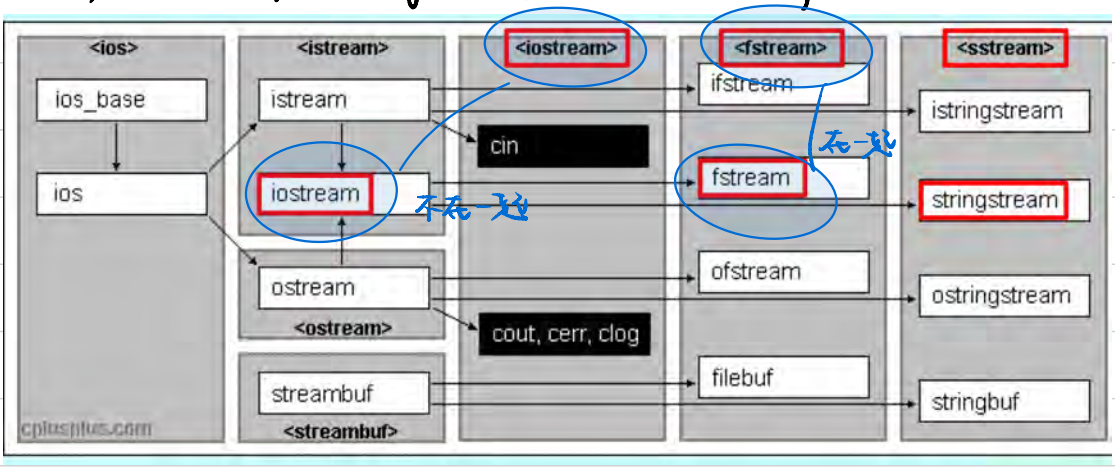
⑤Static variable

	时间上	空间上
Static	全程序	局部 (函数外：全局；函数内：局部)
Global	全程序	全局

2. Library

① Library vs Class Hierarchy

In C++, stream libraries are not organized based on the class hierarchy



3. Creating Library Interface header file

The random.h Interface

```
/*
 * File: random.h
 * --------------
 * This file exports functions for generating pseudorandom numbers.
 */

#ifndef _random_h
#define _random_h

/*
 * Function: randomInteger
 * Usage: int n = randomInteger(low, high);
 * ----------------------------------------
 * Returns a random integer in the range low to high, inclusive.
 */

int randomInteger(int low, int high);
```

是否已复制？

```
/*
 * Function: randomBool
 * Usage: if (randomBool()) ...
 * ----------------------------
 * Returns <code>true</code> with 50% probability.
 */
bool randomBool();

/*
 * Function: setRandomSeed
 * Usage: setRandomSeed(seed);
 * ---------------------------
 * Sets the internal random number seed to the specified value.
 * You can use this function to set a specific starting point
 * for the pseudorandom sequence or to ensure that program
 * behavior is repeatable during the debugging phase.
 */

void setRandomSeed(int seed);

#endif
```

如未复制，#endif 以上都复制

15 Strings

1. Strings in C

①Character

a. Definition

char ⟵ 256 chars in ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Some can be explictly typed and visible, while some may not.

b. Library

cctype (C Char Type)

bool isdigit(char ch) Determines if the specified character is a digit.
bool isalpha(char ch) Determines if the specified character is a letter.
bool isalnum(char ch) Determines if the specified character is a letter or a digit.
bool islower(char ch) Determines if the specified character is a lowercase letter.
bool isupper(char ch) Determines if the specified character is an uppercase letter.
bool isspace(char ch) Determines if the specified character is ***whitespace*** (spaces and tabs).
char tolower(char ch) Converts **ch** to its lowercase equivalent, if any. If not, **ch** is returned unchanged.
char toupper(char ch) Converts **ch** to its uppercase equivalent, if any. If not, **ch** is returned unchanged.

c. Example

```
char ch;
char ch = 'a';
char ch = 97;
```

value: ' ' (单引号是char, 双引号是string)

index: 0-255

不是1-256

String

a. Definition

A sequence of characters (不是Object，是literal)

b. Library

cstring (所有C standard library都以"c"开头)

⚠
```
char cstr[] = "hello";
int len = strlen(cstr);
```
→ 5 (不是6)

⚠ Reassign value

✓
```
strcpy(cstr, "world");
```
小心溢出 cstr是否有足够的空间来复制"world"!

只能strcpy

✗
```
cstr = "world";
cstr[] = "world";
```
We cannot directly assign a value to a C string (not an object)

```
int main() {
    char cstr[] = "hello";
    cout << cstr << endl
         << sizeof cstr << endl
         << strlen(cstr) << endl;
    strcpy(cstr, "hello world");
    cout << cstr << endl
         << sizeof cstr << endl
         << strlen(cstr) << endl;
    return 0;
}
```

6
5
区分size和len

被复制部分 不被复制部分
"hello world"
6 (只计算到原大小)
11 (原理是计算直到"\0")

c. Example

```
char cstr[10];
char cstr[] = "hello";
char cstr[] = { 'h', 'e', 'l', 'l', 'o', '\0' };
```

CString is an array a sequence of basic elements (int, char, bool ...)

length

value: " " (双引号)
'a': 1-char
"a": 2-char
!

value: { } (内部是char，末尾是null char)

2. String in C++

① Definition

A class / An object / An ADT

- C string — 无 method
- C++ string — 有 method

② Library

```
#include <cstring>
// the C string library, used in C++
#include <string.h>
// the C string library, used in C, acceptable in C++
#include "string.h"
// some compilers might still find the C string library, unless you
defined string.h by yourself, which will cause conflict
#include <string>
// the C++ string library
#include "cstring.h"
// incorrect unless you defined cstring.h by yourself
#include <cstring.h>
// most likely an error even if you defined cstring.h by yourself
```

③ Operators

a. Overview

str[i]

+

=

+=

==

str.c_str(): C++ string → C string (array)

str.length()

str.at(index)

str.substr(pos, len): Return the substring

str.find(ch, pos): Return the index

str.find(text, pos): Return the index

str.erase(pos, count)

str.insert(pos, text)

str.replace(pos, count, text)

C++ string is not immutable

b. Further look: Concatenation

Concatenation

- ✓ `cout << "Hello, " << name << "!" << endl;`
- ○ `cout << "Hello, " + name + "!" << endl;`
- ✗ `string str = "hello" + ", " + "world";`
- ✓ `string str = string("hello") + ", " + "world";`
- ✗ `string str = "hello" + ", " + string("world")`

객체 → << cstr, string
\+ string

if name is a cstr: x
if name is a string: ✓

" " → cstr

c. Further look: Length

C : `int len = strlen(str);`

C++ : `int len = str.length();`

L6-7 Streams

1. Streams

① Definition

An ADT used to manage the flow of information to or from some data source

本质: A bridge connecting different data

② Insertion & Extraction operators

- cout << : Insertion
- cin >> : Extraction

③ Input manipulators

- skipws
- noskipws

persistent

- ws

one-time, just for the subsequent input unit.

- Exercise: type in ' ' ' 'A' ' 'B', and what is the output?

```
char a, b, c;
cout << "Enter at least 3 character: ";
cin >> noskipws >> a >> ws >> b >> skipws >> c;
cout << a << b << c << endl;
```

A B

④ Useful methods

Methods supported by all streams

stream`.fail()`	Returns `true` if the stream is in a failure state. This condition usually occurs when you try to read data past the end of the file, but may also indicate an integrity error in the data.
stream`.eof()`	Returns `true` if the stream is positioned at the end of the file. Given the semantics of the C++ stream library, the `eof` method is useful *only* after a call to `fail`. At that point, calling `eof` allows you to test whether the failure indication was caused by reaching the end of the file.
stream`.clear()`	Resets the status bits associated with the stream. You must call this method whenever you need to reuse a stream after a failure has occurred.
`if (`*stream*`) . . .`	Checks whether the stream is valid. For the most part, this test has the same effect as calling `if (!`*stream*`.fail())`.

fail (Eof / error)

Methods supported by all file streams

stream`.open(`*filename*`)`	Attempts to open the named file and attach it to the stream. The direction is determined by the stream type: input streams are opened for input, output streams for output. The *filename* parameter is a C-style string, which means that you will need to call `c_str` on any C++ string. You can check whether the `open` method fails by calling `fail`.
stream`.close()`	Closes the file attached to the stream.

Methods supported by all input streams

stream `>>` *variable*	Reads formatted data into a variable. The data format is controlled by the variable type and whatever input manipulators are in effect.
stream`.get(`*var*`)` 可以改	Reads the next character into the character variable *var*, which is passed by reference. The return value is the stream itself, with the `fail` flag set if there are no more characters.
stream`.get()`	Returns the next character in the stream. The return value is an integer, which makes it possible to identify the end-of-file character, which is represented by the constant `EOF`.
stream`.unget()`	Backs up the internal pointer of the stream so that the last character read will be read again by the next call to `get`.
`getline(`*stream*`,` *str*`)`	Reads the next line of input from *stream* into the string *str*. The `getline` function returns the stream, which simplifies the end-of-file test.

ASCII ID

Methods supported by all output streams

stream `<<` *expression*	Writes formatted data to an output stream. The data format is controlled by the expression type and whatever output manipulators are in effect.
stream`.put(`*ch*`)`	Writes the character *ch* to the output stream.

2. fstream

① Motivation

Data file ⟷ Object (fstream)

② Subclasses

- ifstream
- ofstream

③ Basic procedure

a. Declare
- fstream file
- string filename
- char c

b. Connect file with program

c.
- file.open(...)
- file.get(c)
- edition
- file.put(c)
- file.close()

```
#include <iostream>    cin, cout
#include <fstream>     fstream
#include <string>      string
#include <cctype>      char
using namespace std;
int main() {
a   string filename;
    fstream file;
    char c;
b   cout << "Input file name: ";
    cin >> filename; path
c   file.open(filename.c_str());    . open(Cstring)
    file.get(c); // or c = file.get();
    c = toupper(c);
    file.put(c);
    file.close();
    return 0;
}
```

Does this program uppercase the first character in a file?

Data in files are read and written sequentially

→ 改动要c下一行　　硬件相关 → 机械臂

④ Read chars from a file

```
int main() {
    ifstream infile;
    promptUserForFile(infile, "Input file: ");
    char ch;
    while (infile.get(ch)) {
        cout.put(ch);        cout << ch
    }
    infile.close();
    return 0;
}
```

⑤ Read lines from a file

(lines → string)

```
int max = 0;
string line;
while (getline(infile, line)) {
    if (line.length() > max) max = line.length();
}
```

3. sstream

(...)

28 Collections

1. Overview

① Definition

Collections = Containers = (In this class: Vector, stack, queue, set, map / Not in this class: Array, deque, list...)

Vector is the advance version of array (Vector: finite, controlled / Array: infinite, uncontrolled)

② Different behaviors in two libraries

(Standard lib: 首字母小写 / Stanford lib: 首字母大写)

2. Vector
3. Stack
4. Queue
5. Map
6. Set

→ API看PPT

L9-10 Classes

1. Development: Structure

Structure ⟶ Class

Similarity: Create a type (Class), a variable (Instance)

Difference: Structure +Methods→ Class

- Create a type (Constructor)

```
struct Point {
   int x;
   int y;
};
```

- Create a variable (Instantiate an instance)

```
Point pt;
```

2. Class

① How to write a class

- .h : Interface ⟶ Compile 前被寻找 (include "…h" 的本质是复制粘贴)
- .cpp : Implementation ⟶ Run 时才被寻找

a. .h file

The point.h Interface

```
/*
 * File: point.h
 * -------------
 * This interface exports the Point class, which represents a point
 * on a two-dimensional integer grid.
 */

#ifndef _point_h
#define _point_h

#include <string>

class Point {

public:
```

#ifndef _point_h → 判断是否已"paste" (若是:跳过整个.h文件; 若否:运行至#endif)

#define _point_h → 本质上是个关于"paste"的标记

class Point { → Class内部定义 (Instance variable / Instance method)

The point.h Interface

```
/*
 * Constructor: Point
 * Usage: Point origin;
 *        Point pt(xc, yc);
 * -------------------------
 * Creates a Point object.  The default constructor sets the
 * coordinates to 0; the second form sets the coordinates to
 * xc and yc.
 */

   Point();
   Point(int xc, int yc);
```

The point.h Interface

```
/*
 * Methods: getX, getY
 * Usage: int x = pt.getX();
 *        int y = pt.getY();
 * --------------------------
 * These methods returns the x and y coordinates of the point.
 */

   int getX();
   int getY();

/*
 * Method: toString
 * Usage: string str = pt.toString();
 * ----------------------------------
 * Returns a string representation of the Point in the form "(x,y)".
 * E.g., cout << "pt = " << pt.toString() << endl;
 */

   std::string toString();
```

std::string toString(); → 在.h文件中不要写 using namespace …

The point.h Interface

```
private:

   int x;                    /* The x-coordinate */
   int y;                    /* The y-coordinate */

};

/*
 * Operator: <<
 * Usage: cout << pt;
 * ------------------
 * Overloads the << operator so that it is able to display Point
 * values. E.g., cout << "pt = " << pt << endl;
 */

std::ostream & operator<<(std::ostream & os, Point pt);

#endif
```

Stream不能copy by values, 一定得设置成reference type

#endif → Class外部定义free function (依附非instance method)

.h文件结尾

b. Implementation

The point.cpp Implementation

```
/*
 * File: point.cpp
 * ---------------
 * This file implements the point.h interface.
 */

#include <string>
#include "point.h"
#include "strlib.h"
using namespace std;

/* Constructo
Point::Point() {
   x = 0;
   y = 0;
}

Point::Point(int xc,
   x = xc;
   y = yc;
}
// These two constructors ca  written into
// one, using default arg  nts:
Point(int xc = 0, int y  = 0) {
   x = xc;
   y = yc;
}
// An initializer list can simplify it to:
Point(int xc = 0, int yc = 0):x(xc), y(yc) {};
```

Method definitions are written in exactly the same form as traditional function definitions. The only difference is that you write the name of the class before the name of the method, separated by a double colon.

But this approach does not keep the users from calling it with only one argument.

注意加上namespace

The point.cpp Implementation

```
/* Getters */

int Point::getX() {
   return x;
}

int Point::getY() {
   return y;
}

/* The toString method and the << operator */

string Point::toString() {
   return "(" + integerToString(x) + "," + integerToString(y) + ")";
}

ostream & operator<<(ostream & os, Point pt) {
   retur os << pt.toString();
}
```

If `integerToString()` *is not allowed, how can you do this? Using string streams.*

Why no `Point::` *here? This is a free function.*

因为这是在Point class外部

注意：在class外部定义的free functions无法access private variables

② friend

a. Motivation

If we would like to define free functions inside the class, we can do it by adding "friend" in the header file

```
// in point.h, inside the class Point definition
friend bool operator==(Point p1, Point p2);
// in point.h, outside the class Point definition
bool operator==(Point p1, Point p2);
```

· cpp file不需要

L11-12 Pointers and Arrays

1. Memory

① Bit & Byte & Word & Address

Hotel	Memory
Bed	Bit
Room	Byte = 8 Bits
Floor	Word = 4 Bytes / 8 Bytes
Room number	Address
Number of rooms	Memory size

(char 所用的基本单位) (Smallest addressable unit)

4 Bytes — 32 bit computer

8 Bytes — 64 bit computer (int 所用的基本单位)

多少个 bytes (8GB, 16GB, ...)

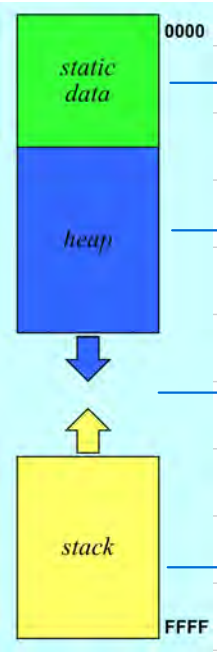
- In theory, N-bit computers can address 2^N bytes of memory

② Size of variable

1 byte (8 bits)	2 bytes (16 bits)	4 bytes (32 bits)	8 bytes (64 bits)	16 bytes (128 bits)
`char` `bool` (实际上只需 1 bit)	`short`	`int` `float`	`long` `double`	`long double`

1 Byte　　1 Word

③ Allocation of memory to variables



static data — Persist throughout the lifetime of the program

heap — Allocate memory dynamically

The stack and heap grow toward each other to maximize the available space

stack — Memory for local variable (Method calling is essentially the stack frame)

(函数 local variable 也叫 automatic variable)

④ lvalue

... can be the left side of an assignment ↔ ... is an lvalue

2. Pointer

① Motivation

Common variable : Does not know the address when being declared

Pointer : Show the address of the variable

② Details

```
int a = 5 ;
int *p = &a;
```

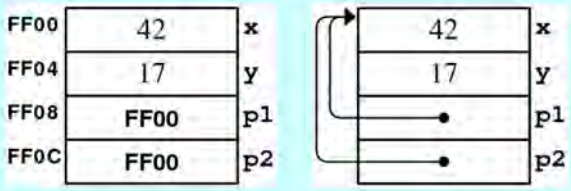
&a 的为地址

*p

- Name : p (不是*p)
- Type : int ← 一直都是int
- Value : Address of a
- Address : In a word

③ Example

```
int x, y;
int *p1, *p2;
x = 42;
y = 163;
p1 = &y;
p2 = &x;
*p1 = 17;
p1 = p2;
*p1 = *p2;
```



p1 = … : 修改 p1 的值 (即改变所指的地址)

*p1 = … : 修改 p1 所指的变量的值

④ Pointer & Reference

Encapsulation

	Pointer	Reference
Definition	The memory address of an object	An alternative identifier for an object
Declaration	`int i = 5;` `int * p = &i;`	`int i = 5;` `int & r = i;`
Dereferencing	`*p`	
Has an address	Yes (`&p`)	No (the same as `&i`)
Pointing/referring to nothing	Yes (`NULL`/ `nullptr` since C++11)	No
Reassignments to new objects	Yes	No
Supported by	C and C++	C++

The address-of operator, not a reference.

(本质上就为另外一个名称)

(指的是 Value)

⑤ Call

Method
- Call by value
- Call by reference
- Call by pointer

a. e.g. for call by reference

```
void swap(int & x, int & y) {
    int tmp = x;
    x = y;
    y = tmp;
}
```

```
swap(n1, n2);
```

b. eg. for call by pointer

```
void swap(int * px, int * py) {
    int tmp = *px;
    *px = *py;
    *py = tmp;
}
```

```
swap(&n1, &n2);
```

3. Array

① Representation

name

```
char cstr[10];
char cstr[] = "hello";
char cstr[] = { 'h', 'e', 'l', 'l', 'o', '\0' };
```

② size

- Allocated size
- Effective size (length)

③ Definition

a. a whole structure — array

b. a pointer to the first element — &array[0]

(the address of the first element)

④ Arithmetic

```
int a[] = {0, 1, 2, 3};
int * p = a; // &a[0]
```

Interpret **a** as an entire array first. If it doesn't make sense, interpret it as a pointer then.

	a	&a	&a[0]	p
Type	array or *used as* pointer to an `int`	address of an array	address of an `int`	pointer to an `int`
Size of	16 (4 `int`)	8 (a word)	8 (a word)	8 (a word)
Lvalue	Yes (non-modifiable)	No	No	Yes
Value	`ADDRESS1`	`ADDRESS1`	`ADDRESS1`	`ADDRESS1`
*	0	`ADDRESS1`	0	0
&	`ADDRESS1`	N/A	N/A	`ADDRESS2`
+1	`ADDRESS1` +1 `int`	`ADDRESS1` +4 `int`	`ADDRESS1` +1 `int`	`ADDRESS1` +1 `int`

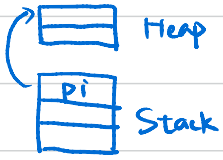
L13-14 Dynamic Memory Management

1. Dynamic Memory Management

① new

```
int * pi = new int;
```

```
type * name = new type[size];
```



② delete

The **delete** operator frees memory previously allocated. For arrays, you need to include empty brackets, as in:

```
delete pi;
delete [] arr;
```

How does the compiler know how much memory to release?

If not delete, the variable will reserve until the end of program (Memory leak)

new → delete

③ Destructor

```
CharStack::CharStack() {
   capacity = INITIAL_CAPACITY;
   array = new char[capacity];
   count = 0;
}

CharStack::~CharStack() {
   delete[] array;
}
```

← Con

← De

④ Copying

a. Two methods

- The ***assignment operator*** has the following form:

```
type & type::operator=(const type & src)
```

Return by reference / *Call by reference*

只能call by reference，因为当copy object时，无法copy by value

- The prototype for the ***copy constructor*** looks like this:

```
type::type(const type & src)
```

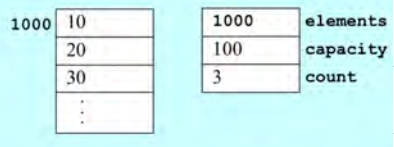
Constructors have the same name as the class (type).

b. Shallow vs Deep Copying

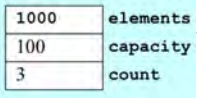
Shallow copying: Default behavior in C++

Deep copying

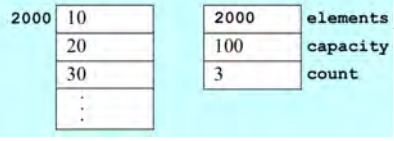
- Suppose that you have an existing `Vector<int>` with three elements as shown in the diagram to the right.



- A ***shallow copy*** allocates new fields for the object itself and copies the information from the original. Unfortunately, the dynamic array is copied as an address, not the data.



- A ***deep copy*** also copies the contents of the dynamic array and therefore creates two independent structures



2. Some expressions

① const

1. ***Constant definitions***. Adding the keyword `const` to a variable definition tells the compiler to disallow subsequent assignments to that variable, thereby making it constant.
```
const double PI = 3.14159265358979323846;
static const int INITIAL_CAPACITY = 10;
```
即使是相同的名

2. ***Constant call by reference***. Adding `const` to the declaration of a reference parameter signifies that the function will not change the value of that parameter. This guarantee allows the compiler to share the contents of a data structure without allowing methods to change it.
```
void deepCopy(const CharStack & src);
```

3. ***Constant methods***. Adding `const` after the parameter list of a method guarantees that the method will not change the object.
```
int CharStack::size() const
```

② static

For the whole class, rather than some certain objects.

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

class A {
public:
   static const int c = 1;
   static int i;
};

int A::i = 2;

int main()
{
   A a, b;
   cout << a.c << a.i << endl;
   cout << b.c << b.i << endl;
   cout << (&a.c == &b.c) << endl;
   cout << (&a.i == &b.i) << endl;
   a.i = 3;
   cout << b.i << endl;
}
```

Now they will have the same address and the same value.

3. Unit Testing

assert(condition)

FIGURE 12-10 Unit test for the CharStack class

```
/*
 * File: CharStackUnitTest.cpp
 * ---------------------------
 * This file contains a unit test of the CharStack class that uses the
 * C++ assert macro to test that each operation performs as it should.
 */

#include <iostream>
#include <cassert>
#include "charstack.h"
using namespace std;

int main() {
   CharStack cstk;                          /* Declare an empty CharStack  */
   assert(cstk.size() == 0);                /* Make sure its size is 0     */
   assert(cstk.isEmpty());                  /* And that isEmpty is true    */
   cstk.push('A');                          /* Push the character 'A'      */
   assert(!cstk.isEmpty());                 /* The stack is now not empty  */
   assert(cstk.size() == 1);                /* And has size 1              */
   assert(cstk.peek() == 'A');              /* Check that peek returns 'A' */
   cstk.push('B');                          /* Push the character 'B'      */
   assert(cstk.peek() == 'B');              /* Make sure peek returns it   */
   assert(cstk.size() == 2);                /* And that the size is now 2  */
   assert(cstk.pop() == 'B');               /* Pop and test for the 'B'    */
   assert(cstk.size() == 1);                /* Recheck the size            */
   assert(cstk.peek() == 'A');              /* And make sure 'A' is on top */
   cstk.push('C');                          /* Test a push after a pop     */
   assert(cstk.size() == 2);                /* Make sure size is correct   */
   assert(cstk.pop() == 'C');               /* And that pop returns a 'C'  */
   assert(cstk.peek() == 'A');              /* The 'A' is now back on top  */
   assert(cstk.pop() == 'A');               /* Pop and test for the 'A'    */
   assert(cstk.size() == 0);                /* And make sure size is 0     */
   for (char ch = 'A'; ch <= 'Z'; ch++) {   /* Push the entire alphabet    */
      cstk.push(ch);                        /*   one character at a time   */
   }                                        /*   to test stack expansion   */
   assert(cstk.size() == 26);               /* Make sure the size is 26    */
   for (char ch = 'Z'; ch >= 'A'; ch--) {   /* Pop the characters in       */
      assert(cstk.pop() == ch);             /*   reverse order to make     */
   }                                        /*   sure they're all there    */
   assert(cstk.isEmpty());                  /* Ensure the stack is empty   */
   for (char ch = 'A'; ch <= 'Z'; ch++) {   /* Push the alphabet again to  */
      cstk.push(ch);                        /*   test that it works after  */
   }                                        /*   expansion                 */
   assert(cstk.size() == 26);               /* Check size is again 26      */
   cstk.clear();                            /* Check the clear method      */
   assert(cstk.size() == 0);                /* And check if stack is empty */
   cstk.clear();                            /* Test clear with empty stack */
   assert(cstk.size() == 0);
   cout << "CharStack unit test succeeded" << endl;
   return 0;
}
```

When false, not stop.

L15-17 Recursion (本章对应见 Lecture Notes)

1. Recursion

① Key component

- Simple case
- Recursive decomposition

本质：数学归纳法

L18 Arithmetic Analysis (详见PP)

L19 Efficiency and Representation

L20 Linear Structures (Implementing Stacks, Queues & Vectors)

1. Template

① Motivation

Generic method

② Method

Incorporate both prototype & implementation in .h file (자료: Ch14-P17-26)

L24 Maps (课本P & PPT)

L22 Tree (講義CP & PPT)

L24 Set

1. Implementation (Using Map)

详见PPT

2. Bitwise Operator

① Content overview

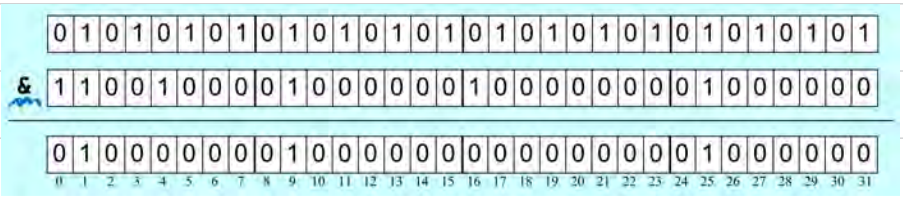
x & y	Bitwise AND. The result has a 1 bit wherever both x and y have 1s.
x | y	Bitwise OR. The result has a 1 bit wherever either x or y have 1s.
x ^ y	Exclusive OR. The result has a 1 bit wherever x and y differ.
~x	Bitwise NOT. The result has a 1 bit wherever x has a 0.
x << n	Left shift. Shift the bits in x left n positions, shifting in 0s.
x >> n	Right shift. Shift x right n bits (logical shift if x is unsigned).

&& Logical operator

& Bitwise operator

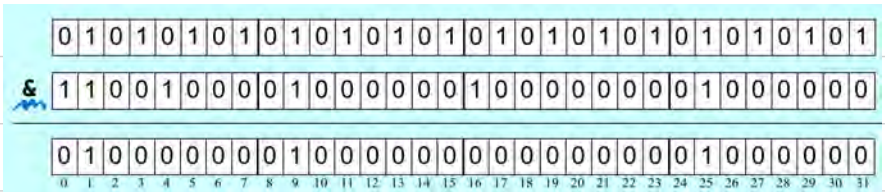
② &

	0	1
0	0	0
1	0	1



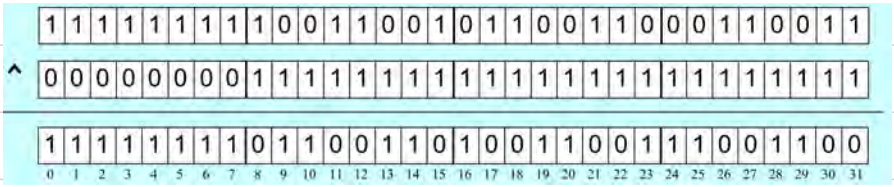
③ |

	0	1
0	0	1
1	1	1



④ ^ 异或

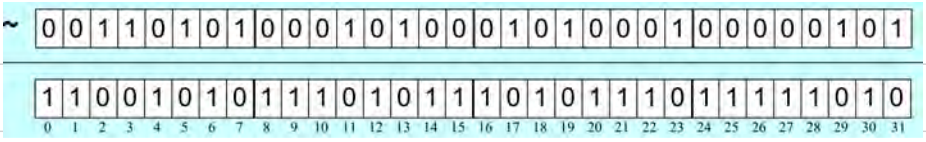
	0	1
0	0	1
1	1	0



(Not both: 1

Both : 0

⑤ ~



思考方式 → 其它进制 → 十进制

⑥Shift (<<, >>) 对于其他数?

left ×√

right ÷√

a. Unsigned (绝对值最大为 $(2^{32}-1)$)

- <<: 移, 补0
- >>:

int 是 signed.

b. Signed

看位

- 0: 正数 (绝对值最大为 $(2^{31}-1)$)
- 1: 负数

①正数

- <<: 移, 补0
- >>: 移, 补0

②负数:

- <<: 移, 补0
- >>: 移, 补1

c. Encoding representation of signed number ⑦现在使用的是?

Two's Complement

- Positive: 和 Original encoding 一样
- Negative: Positive 的 inverse +1 补1

Original: -1: 101

Two's complement: -1: 111

ne digits and adding one.

-bit signed integers:

0	000
1	001
2	010
3	011
−4	100
−3	101
−2	110
−1	111

0	0000 0000
1	0000 0001
2	0000 0010
126	0111 1110
127	0111 1111
−128	1000 0000
−127	1000 0001
−126	1000 0010
−2	1111 1110
−1	1111 1111

a. Exercise

a: 00…0101
b: 00…1010
(a&&b) = true && true → 1
(a&b) = 00…0000 → 0

- What are the outputs:

```
int a = 5;
int b = 10;
cout << (a&&b) << ' ' << (a&b) << endl;
```

```
1 0
```

hexadecimal : 十六進

```
cout << hex << -1 << ' '
<< (-1 << 1) << ' '
<< (-1 >> 1) << ' '
<< unsigned(-1) << ' '
<< (unsigned(-1) << 1) << ' '
<< (unsigned(-1) >> 1) << endl;
```

Largest unsigned integer in 32-bit machines, 2^{32}-1.

Largest positive integer in 32-bit machines, 2^{31}-1.

```
ffffffff fffffffe ffffffff ffffffff ffffffff 7fffffff
-1 -2 -1 4294967295 4294967294 2147483647
```

在 hex 時
在 hex 下

L24 Graph 강의노트 & PPT