Course organization

- **Course introduction (Week 1)**
  - Code editor: Emacs （Week 2）
- **Part I: Introduction to C programming language (Week 3 - 12)**
  - Chapter 1: Overall Introduction (Week 3-4)
  - Chapter 2: Types, operators and expressions (Week 5)
  - Chapter 3: Control flow (Week 6)
  - Chapter 4: Functions and program structure (Week 7)
  - Chapter 5: Pointers and arrays (Week 8)
  - Chapter 6: Structures (Week 10)
  - **Chapter 7: Input and Output (Week 11)**
- **Part II: Skills others than programming languages (Week 12)**
  - Debugging tools（Week 12）
- **Part III: Reports from the battle field (student forum) (Week 13– 16)**
  - Presentation (week 13-14)
  - Demo (week 15-16)

Chapter 7 Input and Output

Chaochun Wei

Shanghai Jiao Tong University

Spring 2019

Contents

- 7.1 Standard input and output
- 7.2 Formatted output -- printf
- 7.3 Variable-length argument lists
- 7.4 Formatted input -- scanf
- 7.5 File access
- 7.6 Error handling -- Stderr and Exit
- 7.7 Line input and output
- 7.8 Miscellaneous Functions

Input and output

- **Not part of the C language itself**
- **They are part of the standard library functions of C**
  - **Standard library functions include**
    - ***Input, output***
    - string handling,
    - storage management
    - Mathematical routines
    - …
  - **They are specified in header files, including**
    - <stdio.h>
    - <string.h>
    - <ctype.h>

7.1 Standard input and output

- Input
  - Read from standard input (keyboard)

    ```
    int getchar(void)
    ```

  - Read characters from a file called infile.

    ```
    prog < infile
    ```

  - Take input from other program otherprog

    ```
    Otherprog | prog
    ```

7.1 Standard input and output

Output

- output to standard output (screen)

```
int putchar(int)
```

- Output to a file outfile

```
Prog > outfile
```

- Output to other program otherprog

```
prog | anotherprog
```

More details see hands-on example 7.1

7.2 Formatted output --printf

- printf
  - syntax of printf

```
int  printf(char *format, arg1, arg2, …)
```

  - Format string
    - Normal characters
    - Conversion characters (begins with a %)
    - A width or precision may be specified as *
  - E.g. , to print at most max characters from a string s:

```
printf(“%.*s”, max, s);
```

More details see hands-on example 7.2 string

7.2 Formatted output --printf

See hands-on example 7.2 number

Format number (%)

Character	Argument type; printed as
d, i	Int; decimal number.
o	Unsigned int; unsigned octal number (without a leading zero)
X, x	Unsigned int; unsigned hexadecima number (without a leading 0x or 0X), using abcdef or ABCDEF for 10,11, 12, 13, 14 and 15.
u	Unsigned int; unsigned decimal number
c	Int; single character.
s	Char *; print a string, until a '\0' or the number of characters given by the precision
f	Double; [-]m.dddddd, where the number of d's is given by the precision (default 6)
e, E	Double; [-]m.dddddd e±xx or [-]m.dddddd E±xx, where the number of d's is given by the precision (default is 6)
p	Void *; pointer (implementation-dependent representation)
%	No argument is converted; print a %

7.3 Variable-length argument lists

- The declaration for printf is

```
int printf(char *fmt, ...)
```

the declaration … means variable-length argument list.

… can only appear at the end of an argument list.

More details see hands-on example 7.3

7.4 Formatted input --scanf

Scanf:

- read characters from the standard input
- Interpret them according to the format string
- Store the results in the remaining arguments

syntax of scanf, sscanf

```
int  scanf(char *format, …)
Int sscanf(char *string, char *format, arg1, arg2, …)
```

Format string

- Blanks or tabs, which are ignored
- Normal characters (not %)
- Conversion characters (begins with a %)

7.4 Formatted input --scanf

Format string (%)

Character	Input data; argument type
d	decimal integer; int *
i	Integer; int *. The integer may be in octal (leading 0) or hexadecimal (leading 0x or 0X)
o	Octal integer(with or without a leading zero); unsigned int *
U	unsigned decimal integer; unsigned int *
x	Hexadecimal integer (with or without leading 0x or 0X); unsigned int *
c	Characters; char *. The next input characters (default 1) are placed at the indicated spot. The normal skip over white space is suppressed; to read the next non-white space character, use %1s.
s	Character string (not quoted); char *, pointing to an array of characters large enough for the string and a terminating '\0' that will be added
e,f,g	Floating-point number with optional sign, optional decimal point and optional exponent; float *
%	No argument is converted; print a %

7.4 Formatted input --scanf

- **The arguments must be pointers in scanf, sscanf**

```
int  scanf(char *format, arg1, arg2, …)
Int sscanf(char *string, char *format, arg1, arg2, …)
```

More details see hands-on example 7.4

7.5 File access

- **Read, write, append**
- **Open a file**

```
FILE *fp;
FILE *fopen(char *name, char *mode);

Mode
  "r": read
  "w": write
  "a": append
  "b": binary files
```

7.5 File access

- **Open a file**

```
FILE *fp;
FILE *fopen(char *name, char *mode);
```

- **Read**
  - If a file does not exist, it's an error
- **Write**
  - If a file does not exist, it will be created
  - If a file exits, the old content will be discarded
- **Append**
  - If a file exits, the old content will be preserved

If there is an error, fopen returns NULL.

7.5 File access

- After a file is open
  - Read the next character from a file
    - int getc (FILE *fp);
  - Write a character t o a file
    - int putc (int c, FILE *fp);

- Close a file after the file access is over
  - *`int fclose(FILE *fp);`*

See more details in hands-on experiment 7.5

7.5 File access

- Formatted input or output of files

```
int fscanf(FILE *fp, char *format, …)

int fprintf(FILE *fp, char *format, …)
```

See more details in hands-on experiment 7.6

7.6 Error handling – stderr and exit

- **When a file can’t be accessed for some reasons**
  - **Stderr: Output the error message on the screen**
  - **Exit: Terminate the program (exit the program)**
    - Terminate the program
    - Close all open output files and
    - Flush out buffered output

See more details in hands-on experiment 7.6

7.7 Line input and output

- Line input
  - *`char *fgets( char *line, int maxline, FILE *fp);`*

    Reads the next input line from file fp into line; at most maxline – 1 characters will be read
- Line output
  - *`int fputs( char *line, FILE *fp);`*

    Writes a string to a file

7.8 Miscellaneous Functions

- String operations : <string.h>

strcat(s, t)	concatenate t to end of s
strncat(s, t, n)	concatenate n characters of t to end of s
strcmp(s, t)	return negative, zero, or positive for s < t, s == t, or s > t
strncmp(s, t, n)	same as strcmp but only in first n chars
strcpy(s, t)	copy t to s
strncpy(s, t, n)	copy at most n characters of t to s
strlen(s)	return length of s
strchr(s, c)	return pointer to first c in s, or NULL if not present
strrchr(s, c)	return pointer to last c in s, or NULL if not present

7.8 Miscellaneous Functions

- Character class testing and conversion

isalpha(c)	non-zero if c is alphabetic, 0 if not
isupper(c)	non-zero if c is upper case, 0 if not
islower(c)	non-zero if c is lower case, 0 if not
isdigit(c)	non-zero if c is digit, 0 if not
isalnum(c)	non-zero if isalpha(c) or isdigit(c), 0 if not
isspace(c)	non-zero if c is blank, tab, newline, return
toupper(c)	return c converted to upper case
tolower(c)	return c converted to lower case

7.8 Miscellaneous Functions

- **Ungetc**
  - ***`int ungetc(int c, FILE *fp);`***

Pushes the character c back into file fp, and returns either c, or EOF for an error.

7.8 Miscellaneous Functions

- Command execution

```
system(char *s);
```

Executes the command contained in string s.

Returns the exit value of command s.

7.8 Miscellaneous Functions

- **Storage management**
  - ***`void *malloc(size_t n);`***
    - Returns a pointer to n bytes of uninitialized storage, or NULL if the request can not be satisfied
  - ***`void *calloc (size_t n, size_t size)`***
    - Returns a pointer to an array of n objects of the specified size, or NULL if failed.
  - ***`void *realloc(void *p, size_t size);`***
    - Changes the size of the object pointed by p to size. Returns a pointer to the new space or NULL if the request can not be satisfied, in which case *p is unchanged

More details see hands-on experiment 7.8

7.8 Miscellaneous Functions

Storage management

- **Type conversion: convert to proper type**

```
int *ip;
ip = (int *) calloc (n, sizeof (int));
```

- **free(p): frees the space pointed to by p, which is obtained by a call to malloc, calloc or realloc**

More details see hands-on experiment 7.8