C/C++ File Operations

한양대학교 컴퓨터소프트웨어학부

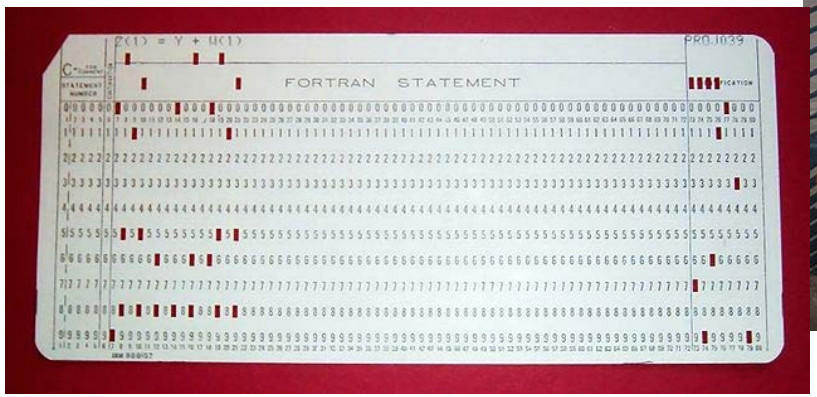
2017년 2학기

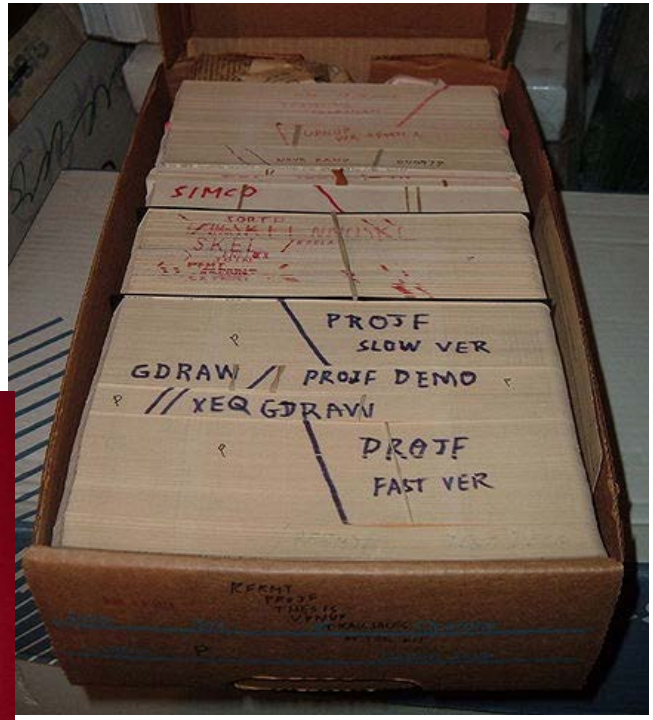
Files

File : a block of arbitrary information, or resource for storing information, which is available to a computer program and is usually based on some kind of durable storage. [wikipedia]

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL





File Properties

- Files are organized into one-dimensional arrays of bytes.
  - The format of a file is defined by its content, since a file is solely a container for data.
  - A file might have a size (number of bytes).
  - File permissions - who may or may not read, modify, delete or create files and folders.
- Files are typically accessed using names, and they can be located in directories.

```
#include <stdio.h>
#include <string.h>
#include <iostream>

int main() {
  FILE* fp_read = stdin;
  FILE* fp_write = stdout;
  if (fp_read == NULL || fp_write == NULL) return -1;
 ...
```

```
hexdump file_op.cc
00000 23 69 6e 63 6c 75 64 65 20 3c 73 74 64 69 6f 2e
00010 68 3e 0a 23 69 6e 63 6c 75 64 65 20 3c 73 74 72
00020 69 6e 67 2e 68 3e 0a 23 69 6e 63 6c 75 64 65 20
00030 3c 69 6f 73 74 72 65 61 6d 3e 0a 0a 69 6e 74 20
00040 6d 61 69 6e 28 29 20 7b 0a 20 20 46 49 4c 45 2a
00050 20 66 70 5f 72 65 61 64 20 3d 20 73 74 64 69 6e
00060 3b 0a 20 20 46 49 4c 45 2a 20 66 70 5f 77 72 69
00070 74 65 20 3d 20 73 74 64 6f 75 74 3b 0a 20 20 69
00080 66 20 28 66 70 5f 72 65 61 64 20 3d 3d 20 4e 55
00090 4c 4c 20 7c 7c 20 66 70 5f 77 72 69 74 65 20 3d
 ...
```

File Operations

- Creating a file with a given name.
- Setting attributes that control operations on the file.
- Opening a file to use its contents.
- Reading or updating the contents.
- Committing updated contents to durable storage.
- Closing the file, thereby losing access until it is opened again.

Unix File Permissions

- Files have owners and groups.
- Read, Write, eXecute permissions for User, Group, Others.
  - Directories : read permission to `ls`, execution to `cd`.
  - `chmod, chown, chgrp`

```
$ ls -al
total 296
drwxr-xr-x   7 jwlim  staff     238 Nov 11 23:23 .
drwxr-xr-x  41 jwlim  staff    1394 Nov 10 16:16 ..
-rwxr-xr-x   1 jwlim  staff  127656 Nov 10 17:06 a.out
-rw-r--r--   1 jwlim  staff    3628 Nov 10 17:06 main.cc
-rw-r--r--   1 jwlim  staff    3593 Nov 10 16:41 main_tmp.cc
-rw-r--r--   1 jwlim  staff    3221 Nov 10 16:17 matrix.cc
-rw-r--r--   1 jwlim  staff    4604 Nov 10 16:47 matrix.h
|\_/\_/\_/     \___/  \___/  \____/ \__________/ \__________/
t u  g  o      owner  group   size    mod. time   file name
y s  r  t
p e  o  h
e r  u  e
     p  r
```

C stdio File Interface - open, close

```c
#include <stdio.h>

int main() {
  // FILE* fopen(const char* filename, const char* mode);
  //   r : read only, r+ : read and write (beginning of the file)
  //   w : truncate and write, w+ : read and write (beginning of the file)
  //   a : write (always end of file), a+ : read and write (always end)
  //   b : binary, ignored.
  FILE* fp = fopen("test.txt", "r");
  if (fp == NULL) return -1;  // Error in opening the file.

  // size_t fread(void* ptr, size_t size, size_t nitems, FILE* stream);
  char buf[2560];
  size_t read = fread(buf, 256, 10, fp);

  fclose(fp);
  return 0;
}
```

C stdio File Interface - read, write

```c
#include <stdio.h>

int main() {
  FILE* fp_read = fopen("source.txt", "r");
  FILE* fp_write = fopen("destination.txt", "w");
  if (fp_read == NULL || fp_write == NULL) return -1;

  char buf[1024];
  size_t read = 0;
  // size_t fread(void* ptr, size_t size, size_t nitems, FILE* stream);
  while ((read = fread(buf, 1, 1024, fp_read)) > 0) {
    // size_t fwrite(const void* ptr, size_t size, size_t nitems,
    //               FILE* stream);
    size_t written = fwrite(buf, read, 1, fp_write);
  }
  fclose(fp_read);
  fclose(fp_write);
  return 0;
}
```

C stdio File Interface - scanf, printf

```c
#include <stdio.h>

int main() {
  FILE* fp_read = fopen("source.txt", "r");
  FILE* fp_write = fopen("destination.txt", "w");
  if (fp_read == NULL || fp_write == NULL) return -1;

  // int fscanf(FILE* stream, const char* format, ...);
  int data;
  while (fscanf(fp_read, "%d", &data) > 0) {
    // int fprintf(FILE* stream, const char* format, ...);
    fprintf(fp_write, "%d\n", data);
  }
  fclose(fp_read);
  fclose(fp_write);
  return 0;
}
```

C stdio File Interface - gets, puts

```c
#include <stdio.h>
#include <string.h>  // memset

int main() {
  FILE* fp_read = fopen("source.txt", "r");
  FILE* fp_write = fopen("destination.txt", "w");
  if (fp_read == NULL || fp_write == NULL) return -1;

  // char* fgets(char* str, int size, FILE* stream);
  char buf[1024];
  memset(buf, 0, 1024);
  while (fgets(buf, 1023, fp_read) > 0) {
    // int fputs(const char* str, FILE* stream);
    fputs(buf, fp_write);
  }
  fclose(fp_read);
  fclose(fp_write);
  return 0;
}
```

C stdio File Interface - end of file

```
#include <stdio.h>
#include <string.h>  // memset
#include <iostream>

int main() {
  FILE* fp = fopen("test.txt", "r");
  if (fp == NULL) return -1;  // Error in opening the file.

  char buf[1024];
  memset(buf, 0, 1024);
  // int feof(FILE* stream); - non-zero if end-of-file flag is set.
  while (!feof(fp)) {
    fgets(buf, 1023, fp);
    std::cout << buf;
  }

  fclose(fp);
  return 0;
}
```

C stdio File Interface - seek

```
#include <stdio.h>
#include <iostream>

int main() {
  FILE* fp = fopen("data.bin", "r");
  if (fp == NULL) return -1;  // Error in opening the file.

  // int fseek(FILE* stream, long offset, int whence);
  //   SEEK_SET, SEEK_CUR, SEEK_END; returns 0 if successful.
  // void rewind(FILE* stream);  = fseek(stream, 0L, SEEK_SET);
  // long ftell(FILE* stream);
  fseek(fp, 0L, SEEK_END);
  std::cout << ftell(fp) << std::endl;  // Prints the size of the file.

  char buf[256];
  fseek(fp, 1024L, SEEK_SET);
  size_t read = fread(buf, 256, 10, fp);

  fclose(fp);
  return 0;
}
```

C stdio File Interface - stdin, stdout

```c
#include <stdio.h>
#include <string.h>  // memset

int main() {
  FILE* fp_read = stdin;
  FILE* fp_write = stdout;
  if (fp_read == NULL || fp_write == NULL) return -1;

  char buf[1024];
  memset(buf, 0, 1024);
  // char* fgets(char* str, int size, FILE* stream);
  while (fgets(buf, 1023, fp_read) > 0) {
    // int fputs(const char* str, FILE* stream);
    fputs(buf, fp_write);
  }
  fclose(fp_read);
  fclose(fp_write);
  return 0;
}
```

C++ File Stream Interface

- `ifstream`, `ofstream` : similar to cin, cout, but for files.

```
// print the content of a text file.
#include <iostream>
#include <fstream>

using namespace std;

int main () {
  ifstream infile;
  infile.open("test.txt", ifstream::in);

  int ch = infile.get();
  while (infile.good()) {
    cout << (char) ch;
    ch = infile.get();
  }
  infile.close();
  return 0;
}
```

Streams

- A flow of characters
- Input stream
  - Flow into program
    - Can come from keyboard
    - Can come from file
- Output stream
  - Flow out of program
    - Can go to screen
    - Can go to file

Streams Usage

- We've used streams already
  - cin
    - Input stream object connected to keyboard
  - cout
    - Output stream object connected to screen
- Can define other streams
  - To or from files
  - Used similarly as cin, cout

Streams Usage Like cin, cout

Consider:

- Given program defines stream inStream that comes from some file:

```
int theNumber;
inStream >> theNumber;
```

  - Reads value from stream, assigned to *theNumber*

- Program defines stream outStream that goes to some file

```
outStream << "theNumber is " << theNumber;
```

  - Writes value to stream, which goes to file

Files

- We'll use text files
- Reading from file
  - When program takes input
- Writing to file
  - When program sends output
- Start at beginning of file to end
  - Other methods available
  - We'll discuss this simple text file access here

File Connection

- Must first connect *file* to *stream object*
- For input:
  - File → ifstream object
- For output:
  - File → ofstream object
- Classes ifstream and ofstream
  - Defined in library <fstream>
  - Named in std namespace

File I/O Libraries

- To allow both file input and output in your program:

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

OR

```
#include <fstream>
using std::ifstream;
using std::ofstream;
```

Declaring Streams

- Stream must be declared like any other class variable:

```
ifstream inStream;
ofstream outStream;
```

- Must then "connect" to file:

```
inStream.open("infile.txt");
```

  - Called "opening the file"
  - Uses member function *open*
  - Can specify complete pathname

Streams Usage

- Once declared → use normally!

```
int oneNumber, anotherNumber;
inStream >> oneNumber >> anotherNumber;
```

- Output stream similar:

```
ofstream outStream;
outStream.open("outfile.txt");
outStream      << "oneNumber = " << oneNumber
               << " anotherNumber = "
               << anotherNumber;
```

  - Sends items to output file

File Names

- Programs and files
- Files have two names to our programs
  - External file name
    - Also called "physical file name"
    - Like "infile.txt"
    - Sometimes considered "real file name"
    - Used only once in program (to open)
  - Stream name
    - Also called "logical file name"
    - Program uses this name for all file activity

Closing Files

- Files should be closed
  - When program completed getting input or sending output
  - Disconnects stream from file
  - In action:

    ```
    inStream.close();
    outStream.close();
    ```

    - Note no arguments
- Files automatically close when program ends

File Flush

- Output often "buffered"
  - Temporarily stored before written to file
  - Written in "groups"
- Occasionally might need to force writing: outStream.flush();
  - Member function *flush*, for all output streams
  - All buffered output is physically written
- Closing file automatically calls flush()

Display 12.1 Simple File Input/Output

```
1  //Reads three numbers from the file infile.txt, sums the numbers,
2  //and writes the sum to the file outfile.txt.
3  #include <fstream>
4  using std::ifstream;
5  using std::ofstream;
6  using std::endl;

7  int main( )
8  {
9      ifstream inStream;
10     ofstream outStream;

11     inStream.open("infile.txt");
12     outStream.open("outfile.txt");

13     int first, second, third;
14     inStream >> first >> second >> third;
15     outStream << "The sum of the first 3\n"
16               << "numbers in infile.txt\n"
17               << "is " << (first + second + third)
18               << endl;
```

A better version of this program is given in Display 12.3.

```
19          inStream.close( );
20          outStream.close( );

21          return 0;
22      }
```

SAMPLE DIALOGUE

There is no output to the screen
and no input from the keyboard.

infile.txt
(Not changed by program)

```
1
2
3
4
```

outfile.txt
(After program is run)

```
The sum of the first 3
numbers in infile.txt
is 6
```

Appending to a File

- Standard open operation begins with empty file
  - Even if file exists → contents lost
- Open for append:

```
ofstream outStream;
outStream.open("important.txt", ios::app);
```

  - If file doesn't exist → creates it
  - If file exists → appends to end
  - 2nd argument is class *ios* defined constant
    - In <iostream> library, std namespace

Alternative Syntax for File Opens

- Can specify filename at declaration
  - Passed as argument to constructor
- ifstream inStream;
  inStream.open("infile.txt");

  EQUIVALENT TO:

  ifstream inStream("infile.txt");

Checking File Open Success

- File opens could fail
  - If input file doesn't exist
  - No write permissions to output file
  - Unexpected results
- Member function fail()
  - Place call to fail() to check stream operation success

```
inStream.open("stuff.txt");
if (inStream.fail())
{
    cout << "File open failed.\n";
    exit(1);
}
```

Character I/O with Files

- All cin and cout character I/O same for files!
- Member functions work same:
  - get, getline
  - put, putback,
  - peek, ignore

Checking End of File

- Use loop to process file until end
  - Typical approach
- Two ways to test for end of file
  - Member function eof()

```
inStream.get(next);
while (!inStream.eof())
{
    cout << next;
    inStream.get(next);
}
```

    - Reads each character until file ends
    - eof() member function returns bool

End of File Check with Read

Second method

- read operation returns bool value!
  (inStream >> next)
  - Expression returns true if read successful
  - Returns false if attempt to read beyond end of file
- In action:

```
double next, sum = 0;
while (inStream >> next)
    sum = sum + next;
cout << "the sum is " << sum << endl;
```

Random Access to Files

- Sequential Access
  - Most commonly used
- Random Access
  - Rapid access to records
  - Perhaps very large database
  - Access "randomly" to any part of file
  - Use fstream objects
    - input and output

Random Access Tools

- Opens same as istream or ostream
  - Adds second argument
  - fstream rwStream;
    rwStream.open("stuff", ios::in | ios:: out);
    - Opens with read and write capability
- Move about in file
  - rwStream.seekp(1000);
    - Positions put-pointer at 1000th byte
  - rwStream.seekg(1000);
    - Positions get-pointer at 1000th byte

Random Access Sizes

- To move about → must know sizes
  - sizeof() operator determines number of bytes required for an object:
    sizeof(s) //Where s is string s = "Hello"
    sizeof(10)
    sizeof(double)
    sizeof(myObject)
  - Position put-pointer at 100th record of objects:

    rwStream.seekp(100*sizeof(myObject) – 1);

C unistd File Interface

Used when low-level control on files is needed.

e.g. change permission, network socket communication, file locking, etc.

- Headers : `unistd.h`, `fcntl.h`
- `FILE* fp` : `int file_descriptor`
- `fopen`, `fclose`, `fread`, `fwrite` , … :

  `open`, `close`, `read`, `write`, …


Q&A