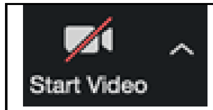
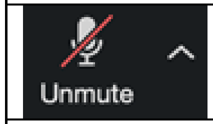
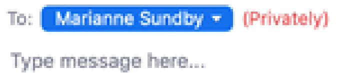
Denne forelesningsøkten vil bli tatt opp og lagt ut i emnet i etterkant.

Hvis du ikke vil være med på opptaket:

	La være å delta med webkameraet ditt.
	La være å delta med mikrofonen din.
 To: Marianne Sundby (Privately) Type message here...	Still spørsmål i Chat i stedet for som lyd. Hvis du ønsker kan spørsmålet også sendes privat til foreleser.

Høyskolen
Kristiania

Høyskolen Kristiania

PG3401 Programmering i C for Linux

Bengt Østby

Recap

- Arrays
- String
- Structs

Multidimensional arrays – repetition

Static arrays can be declared easily :

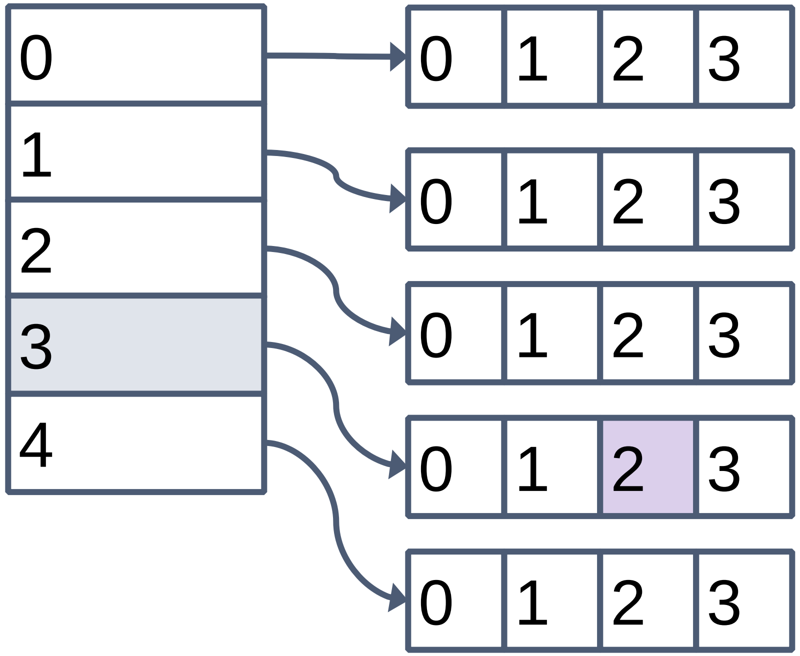
```
int iArr1[5];
int *piArr1;

int iArr2[5][4];
int **piArr2;
```

Accessing elements :

```
piArr2 = iArr2; // i = piArr2[r][c];
piArr2[r]    // Access row == pointer to row
piArr2[r][c] // Access col ⇔ iArr[r][c]
```

Multidimensional arrays and pointers

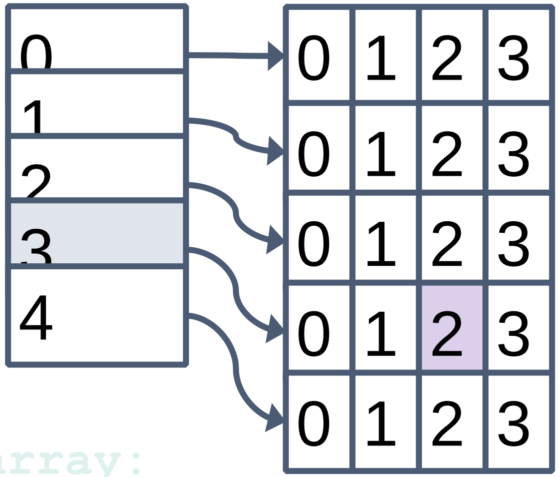
```
int iArr2[5][4];
```



```
int **piArr2 = iArr2;
int *piArr3 = piArr2[3];
int i = piArr3[2]; // Same as iArr2[3][2]
```

Multidimensional arrays – dynamically allocated

```
int *piArr2;
```

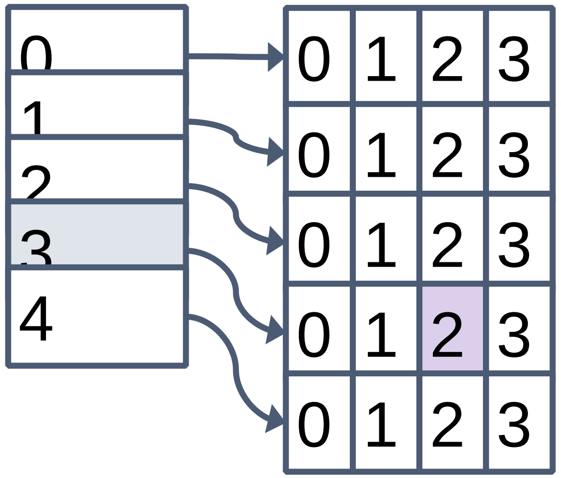


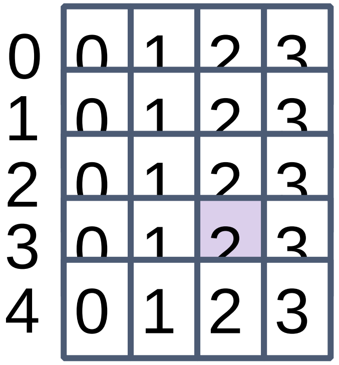
```
iRows = 5; iCols = 4;
// Allocate first the rows as an int* array:
piArr2 = (int **) malloc (iRows * sizeof(int *));
for (int i = 0; i < iRows; i++) {
  // For each row, allocate the columns as int arrays:
  piArr2[i] = (int *) malloc (iCols * sizeof(int));
}
```

```
int *piArr3 = piArr2[3];
int i = piArr3[2];
```

Multidimensional arrays #4 – the C11 way

```
char acArr1[5][4];
char acArr2[5*4];

sizeof(acArr1)
sizeof(acArr2) ??
```





I/O

- Important to talk to the program!
- Or let the program talk to us!
- Or let programs talk to each other

Standard I/O

stdio.h – library for I/O in C

Flexibility in I/O

linux + C = powerful!

Input to program

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

argc – number of arguments

argv – array of strings

Example :

- $ ./myprog ar1 ar2 ar3
- argc is 4
- argv[0] = ./myprog
- argv[1] = ar1
- argv[2] = ar2
- argv[3] = ar3
- argv[4] = *who knows*

I/O during run-time

Standard Input – usually the keyboard

Standard Output – usually the display

C also supports file I/O

Redirection

‘<‘ and ‘>’ can be used to redirect standard output/input to files.

Output redirection is trivial!

Example :

```
#include <stdio.h>

int main(int argc, char *argv[]){
    int a=42;
    printf("Hi this is just an output\n");
    printf("It supports %d formats", a);
    return 0;
}
```

Example redirection:

- $ ./a.out > randomFile

Terminal

int putchar(int)

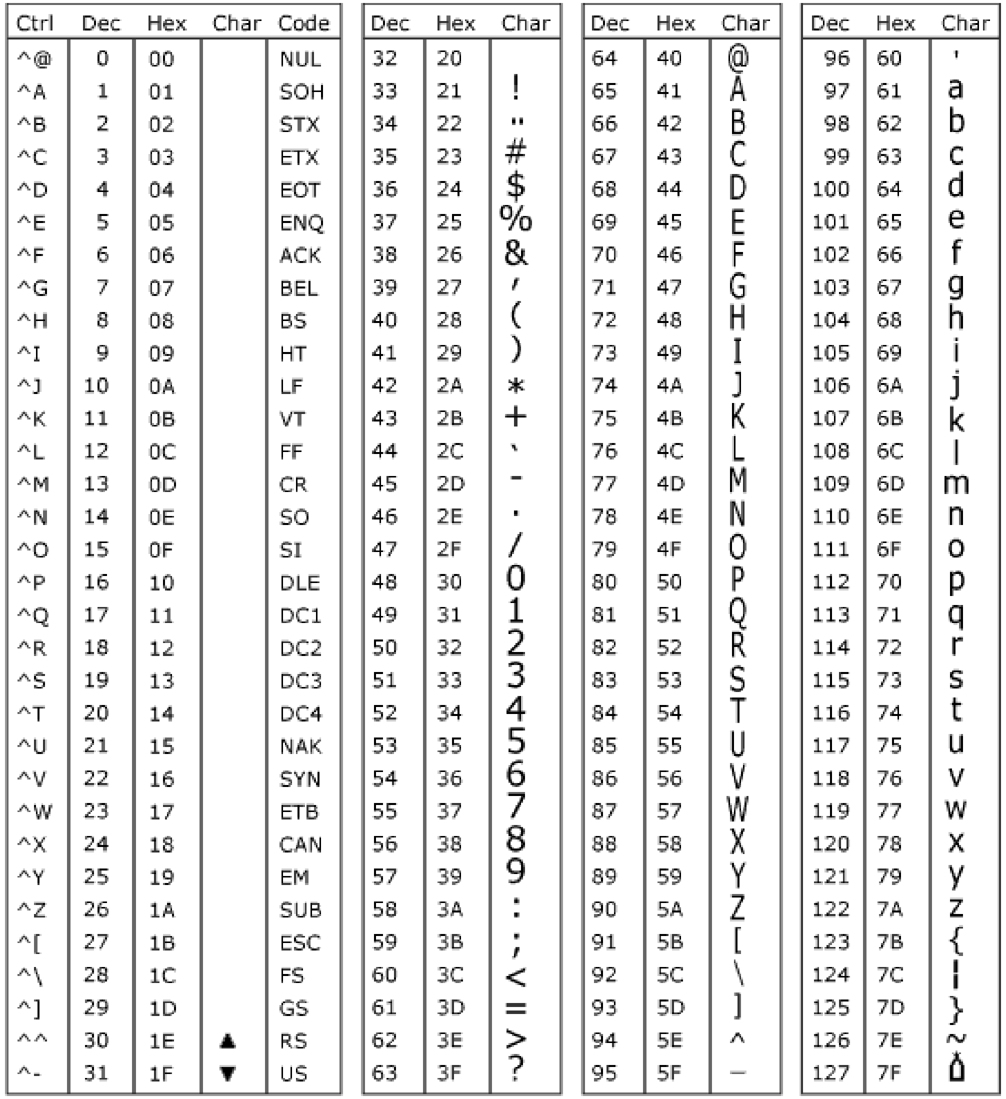
- puts a character to the *standard* output
- returns the character or EOF on error

int getchar()

- a ***blocking*** input for a character
- returns the ASCII-value of next character from the *standard* input
- returns **EOF** on error

- @ is 64, A is 65, a is 97
- Google ASCII character table
- printf (“as char %c as number %d”, 65, ‘A’);
  → as character A as number 65.

ASCII character table



Ctrl	Dec	Hex	Char	Code	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
^@	0	00		NUL	32	20		64	40	@	96	60	`
^A	1	01		SOH	33	21	!	65	41	A	97	61	a
^B	2	02		STX	34	22	"	66	42	B	98	62	b
^C	3	03		ETX	35	23	#	67	43	C	99	63	c
^D	4	04		EOT	36	24	$	68	44	D	100	64	d
^E	5	05		ENQ	37	25	%	69	45	E	101	65	e
^F	6	06		ACK	38	26	&	70	46	F	102	66	f
^G	7	07		BEL	39	27	'	71	47	G	103	67	g
^H	8	08		BS	40	28	(	72	48	H	104	68	h
^I	9	09		HT	41	29	)	73	49	I	105	69	i
^J	10	0A		LF	42	2A	*	74	4A	J	106	6A	j
^K	11	0B		VT	43	2B	+	75	4B	K	107	6B	k
^L	12	0C		FF	44	2C	,	76	4C	L	108	6C	l
^M	13	0D		CR	45	2D	-	77	4D	M	109	6D	m
^N	14	0E		SO	46	2E	.	78	4E	N	110	6E	n
^O	15	0F		SI	47	2F	/	79	4F	O	111	6F	o
^P	16	10		DLE	48	30	0	80	50	P	112	70	p
^Q	17	11		DC1	49	31	1	81	51	Q	113	71	q
^R	18	12		DC2	50	32	2	82	52	R	114	72	r
^S	19	13		DC3	51	33	3	83	53	S	115	73	s
^T	20	14		DC4	52	34	4	84	54	T	116	74	t
^U	21	15		NAK	53	35	5	85	55	U	117	75	u
^V	22	16		SYN	54	36	6	86	56	V	118	76	v
^W	23	17		ETB	55	37	7	87	57	W	119	77	w
^X	24	18		CAN	56	38	8	88	58	X	120	78	x
^Y	25	19		EM	57	39	9	89	59	Y	121	79	y
^Z	26	1A		SUB	58	3A	:	90	5A	Z	122	7A	z
^[	27	1B		ESC	59	3B	;	91	5B	[	123	7B	{
^\	28	1C		FS	60	3C	<	92	5C	\	124	7C	¦
^]	29	1D		GS	61	3D	=	93	5D	]	125	7D	}
^^	30	1E	▲	RS	62	3E	>	94	5E	^	126	7E	~
^-	31	1F	▼	US	63	3F	?	95	5F	_	127	7F	⌂*

* ASCII code 127 has the code DEL. Under MS-DOS, this code has the same effect as ASCII 8 (BS). The DEL code can be generated by the CTRL + BKSP key.

printf()

int printf(…)

The first argument is a string that holds the formatting pattern.

- Following arguments should make sense!
- Compiler will warn you!
- The number of arguments is basically your responsibility!
- returns the number of characters printed
- return value rarely checked
- More intuitive than java’s output mechanism?

snprintf() and fprintf()

snprintf – printf to a string

snprintf (char *str, size_t size, char * format, ...);

fprintf – printf to file

fprintf (FILE *stream, char *format, ...);

Formatted string

The format is the following :

- %[parameter][flags][width][.precision][length]type
- parameter – n$ - used to specify the number of argument
- flags - +, <space>, …, leading zeroes
- width – *minimum* number of chars
- precision – *maximum* number after . or chars
- length – deciding the length
- type – type of the argument
- * - to pass the width and precision as arguments

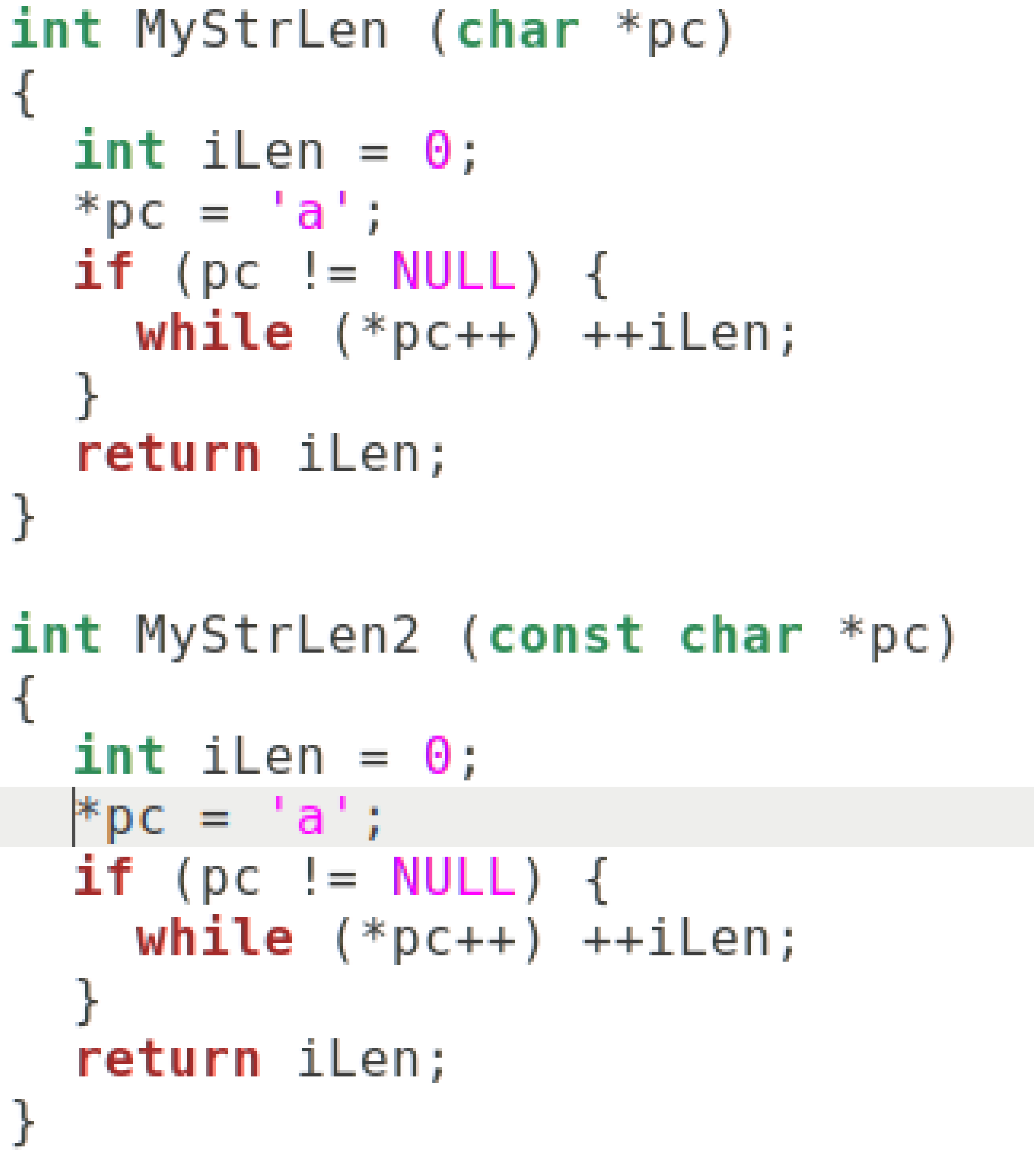
How do I print % ?

scanf()

int scanf(…)

- Much like printf() but for input from standard input
- scanf() needs a *pointer* to where to store the values read
- returns the number of items read
- The return value must be checked for a safe execution!
- matches all the formatted string to the input
  - Example : scanf(“%d, %d”);

const ??



```
int MyStrLen (char *pc)
{
  int iLen = 0;
  *pc = 'a';
  if (pc != NULL) {
    while (*pc++) ++iLen;
  }
  return iLen;
}

int MyStrLen2 (const char *pc)
{
  int iLen = 0;
  *pc = 'a';
  if (pc != NULL) {
    while (*pc++) ++iLen;
  }
  return iLen;
}
```

Files

Everything is a file on linux!

Steps in handling a file:

- Open the file
- Perform all the I/O
- Close the file

Proper opening and closing of file are important!

Typically two types :

- text – usually line oriented
- binary

See https://www.tutorialspoint.com/c_standard_library/stdio_h.htm

FILE *

- Pointer to the type FILE
  - Which is a file handle!
  - Usually passed into library functions
  - Hence passed as pointer

Opening a file

FILE *fopen (**const** char *filename, **const** char *mode)

- opens file and returns a file handle
- first argument is a file name
- mode can be
  - r Open for read. File must exist.
  - w Open for write. Existing file will be truncated.
  - a Append. Everything written will be appended.
  - r+ Read and write. The file must exist.
  - w+ Read and write. Existing file will be truncated.
  - a+ Read and append. The file must exist.
- suffix ‘b’ for binary files
- returns NULL on failure.
  (Check global int errno for reason.)

Writing a file

int fputc(int c, FILE *f)

int fputs(const char *s, FILE *f)

int fprintf(FILE *f, ……)

Reading from a file

```
int fgetc(FILE *f)
char *fgets(char *buf, int n, FILE *f)
char *fscanf(FILE *f, …)
```

EOF & EOL

In C ‘\n’ is end-of-line

EOF is end-of-file – actually a character.

EOF & EOL

```
FILE *f = NULL;
char szLine [160]; // Assume max line length
int iLine = 0;

f = fopen ("test.txt", "r"); // Text read.
if (f != NULL) {
   while (!feof(f)) {
      if (fgets (szLine, sizeof(szLine), f)) {
         printf ("%3d: %s", ++iLine, szLine);
      }
   }
   fclose (f);
}
```

Buffered input/output

the i/o are always buffered

```
int ungetc(int c, FILE *f)
int fflush(FILE *f)
int fseek(FILE *f, long int p, int o)
int fsetpos(FILE *f, const fpos_t *p)
int fgetpos(FILE *f, fpos_t *p)
fpos_t ftell(FILE *f)
void rewind(FILE *f)
FILE *tmpfile()
```

Closing a file

int fclose(FILE *f)

All the buffers will be flushed, so an important call

returns EOF on failure else 0

Be careful, if error then pretty much everything is lost!

Binary files

why binary files?

Example :

- double values = ‘8901928.7381029’ – 15 bytes as text!
- communication between programs

Treat like text files but the input and output are byte streams

Opening flag is suffixed with ‘b’

- Example : fopen(“some/binary/file.mp3”, “rb”);

Not strictly platform independent –

- Big Endian
- Little Endian

Reading binary files

size_t fread(void *buf, size_t size, size_t count, FILE *f)

- buf is the place to store your data
- size is the size of each element
- count is the number of elements to be read
- f is the file handle

returns the number of elements read (should be count)

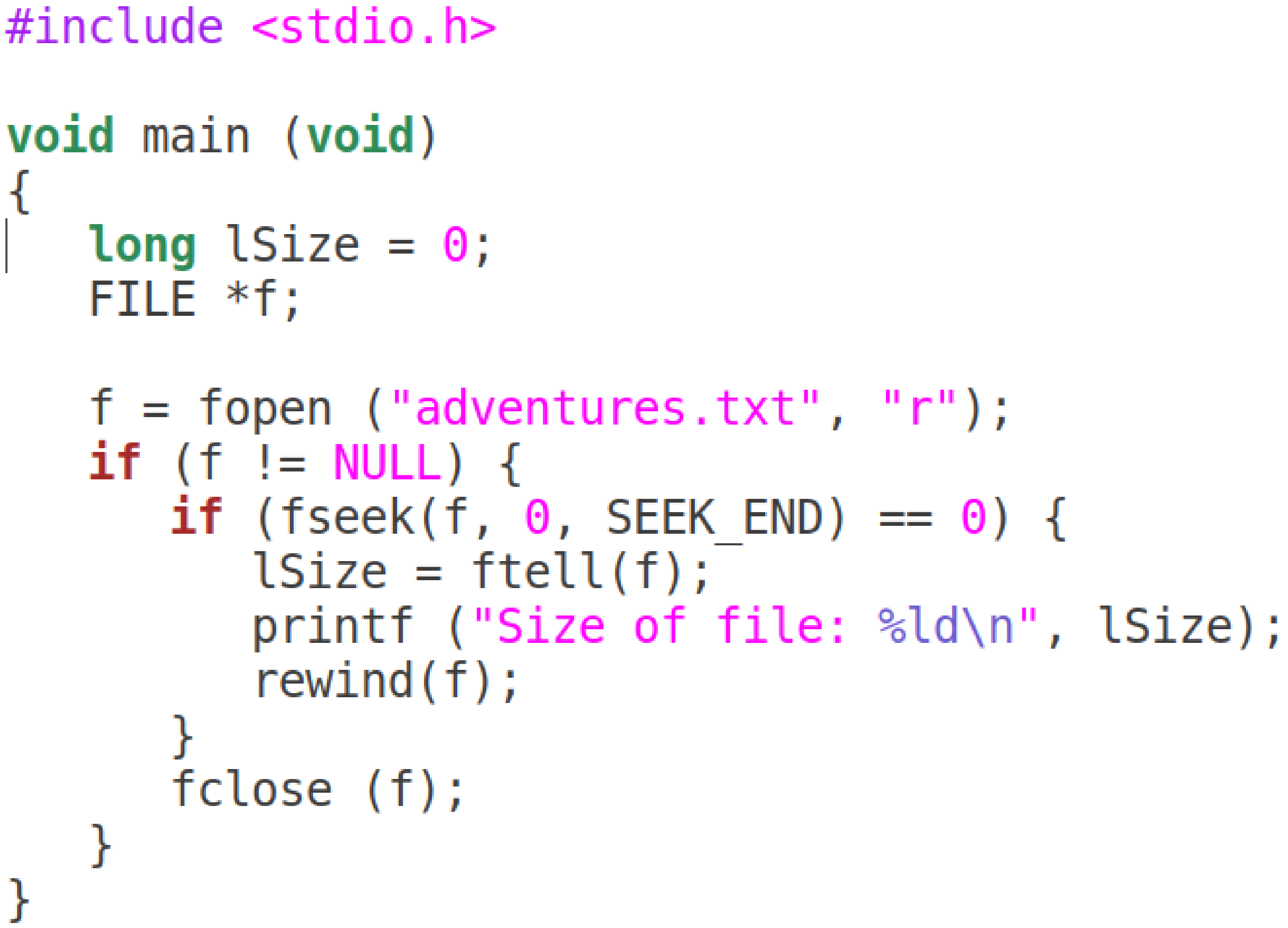
Writing to binary files

size_t fwrite(const void *ptr, size_t size, size_t count, FILE *f)

- ptr – data source
- size – size of each element
- count – number of elements
- f - file handle

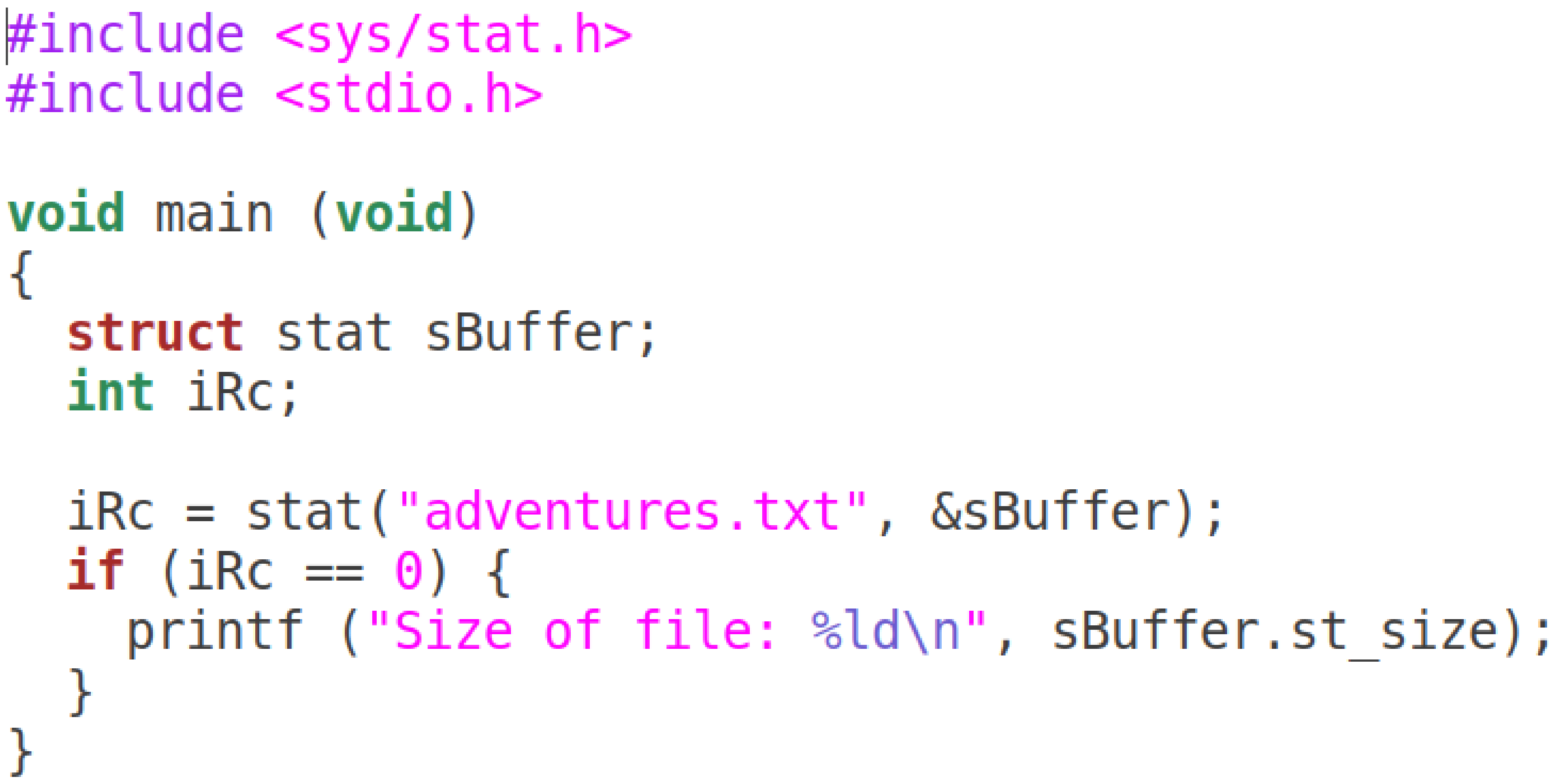
Returns number of elements written

Getting the size of a file…



```c
#include <stdio.h>

void main (void)
{
   long lSize = 0;
   FILE *f;

   f = fopen ("adventures.txt", "r");
   if (f != NULL) {
      if (fseek(f, 0, SEEK_END) == 0) {
         lSize = ftell(f);
         printf ("Size of file: %ld\n", lSize);
         rewind(f);
      }
      fclose (f);
   }
}
```

Getting the size of a file…



```
#include <sys/stat.h>
#include <stdio.h>

void main (void)
{
  struct stat sBuffer;
  int iRc;

  iRc = stat("adventures.txt", &sBuffer);
  if (iRc == 0) {
    printf ("Size of file: %ld\n", sBuffer.st_size);
  }
}
```

See also http://www.cplusplus.com/reference/cstdio/

Named Pipe

Essentially a FIFO

unidirectional data channel

Simply a file on your machine

Example :

- mkfifo myPipe
- See https://linuxprograms.wordpress.com/tag/mkfifo/
- https://www.softprayog.in/programming/interprocess-communication-using-fifos-in-linux

Working with pipes

just as any files!

blocking I/O

- Must be handled at both ends

Typically used for using third-party programs and asynchronous communication between applications

Exercises

```
wget www.eastwillsecurity.com/pg3401/leksjon5.zip
```