**#include <unistd.h>**

**#include <stdarg.h>**

**--------------------------------------------------------------**

**int ft\_putchar(int c)**

**{**

**write(1, &c, 1);**

**return (1);**

**}**

**--------------------------------------------------------------**

**int ft\_putstr(char \*str)**

**{**

**int count;**

**if (!str)**

**{**

**write(1, "(null)", 6);**

**return (6);**

**}**

**count = 0;**

**while (\*str)**

**{**

**count += ft\_putchar(\*str);**

**str++;**

**}**

**return (count);**

**}**

**--------------------------------------------------------------**

**int ft\_putnbr(int n)**

**{**

**int count;**

**if (n == -2147483648)**

**{**

**ft\_putstr("-2147483648");**

**return (11);**

**}**

**count = 0;**

**if (n < 0)**

**{**

**n \*= -1;**

**count += ft\_putchar('-');**

**}**

**if (n > 9)**

**count += ft\_putnbr(n / 10);**

**count += ft\_putchar((n % 10) + '0');**

**return (count);**

**}**

**--------------------------------------------------------------**

**int ft\_putuhexnbr(unsigned int n)**

**{**

**int count;**

**count = 0;**

**if (n >= 16)**

**count += ft\_putuhexnbr(n / 16);**

**count += ft\_putchar("0123456789abcdef"[n % 16]);**

**return (count);**

**}**

**--------------------------------------------------------------**

**int ft\_printf(const char \*str, ...)**

**{**

**int     count;**

**va\_list ap;**

**count = 0;**

**va\_start(ap, str);**

**while (\*str)**

**{**

**if (\*str == '%')**

**{**

**str++;**

**if (\*str == 's')**

**count += ft\_putstr(va\_arg(ap, char \*));**

**else if (\*str == 'd')**

**count += ft\_putnbr(va\_arg(ap, int));**

**else if (\*str == 'x')**

**count += ft\_putuhexnbr(va\_arg(ap, unsigned int));**

**}**

**else**

**count += ft\_putchar(\*str);**

**str++;**

**}**

**va\_end(ap);**

**return (count);**

**}**

/\*

**Assignment name : get\_next\_line**

Expected files : get\_next\_line.c get\_next\_line.h

Allowed functions: read, free, malloc

Write a function named get\_next\_line which prototype should be:

char \*get\_next\_line(int fd);

Your function must return a line that has been read from the file descriptor passed as parameter.

What we call a "line that has been read" is a succesion of 0 to n characters that end with '\n'

(ascii code 0x0a) or with End Of File (EOF). The line should be returned including the '\n' in case

there is one at the end of the line that has been read. When you've reached the EOF, you must store

the current buffer in a char \* and return it.

If the buffer is empty you must return NULL.

In case of error return NULL.

In case of not returning NULL, the pointer should be free-able.

Your program will be compiled with the flag -D BUFFER\_SIZE=xx, which has to be used as the buffer size

for the read calls in your functions. Your function must be memory leak free. When you've reached the EOF,

your function should keep 0 memory allocated with malloc, except the line that has been returned.

Caling your function get\_next\_line in a loop will therefore allow you to read the text avaiable on

a file descriptor one line at a time until the end of the text, no matter the size of either the text or one of its lines.

Make sure that your function behaves well when it reads from a file, from the standard output, from a redirection, etc...

No call to another function will be done on the file descriptor between 2 calls of get\_next\_line.

Finally we consider that get\_next\_line has an undefined behaviour when reading from a binary file.

\*/

/\*

**Assignment name : ft\_printf**

Expected files : ft\_printf.c

Allowed functions: malloc, free, write, va\_start, va\_arg, va\_copy, va\_end

Write a function named ft\_printf that will mimic the real printf but it will manage only the following conversions:

s (string), d (decimal) and x (lowercase hexademical).

Your function must be declared as follows:

    int ft\_printf(const char \*, ... );

Examples of the function output:

    call: ft\_printf("%s\n", "toto"); out: toto$

    call: ft\_printf("Magic %s is %d", "number", 42); out: Magic number is 42%

    call: ft\_printf("Hexadecimal for %d is %x\n", 42, 42); out: Hexadecimal for 42 is 2a$

\*/

**get\_next\_line.c**

#include "get\_next\_line.h"

#define FD\_LIMIT 1024

#ifndef BUFFER\_SIZE

# define BUFFER\_SIZE 42

#endif

#if (BUFFER\_SIZE > 1000000)

# undef BUFFER\_SIZE

# define BUFFER\_SIZE 1000000

#endif

**--------------------------------------------------------------**

int ft\_strlen(char \*str)

{

int len;

if (!str)

return (0);

len = 0;

while (\*str++)

len++;

return (len);

}

**--------------------------------------------------------------**

int ft\_isline(char \*str)

{

if (!str)

return (0);

while (\*str)

{

if (\*str == '\n')

return (1);

str++;

}

return (0);

}

**--------------------------------------------------------------**

int ft\_get\_line\_len(char \*str)

{

int i;

i = 0;

while (\*(str + i))

{

if (\*(str + i++) == '\n')

return (i);

}

return (i);

}

**--------------------------------------------------------------**

char \*ft\_strjoin(char \*s1, char \*s2)

{

size\_t i;

size\_t j;

char \*join;

if (s1 == NULL || s2 == NULL)

return (NULL);

join = (char \*)malloc(ft\_strlen(s1) + ft\_strlen(s2) + 1);

if (join == NULL)

return (NULL);

i = 0;

while (\*(s1 + i))

{

\*(join + i) = \*(s1 + i);

i++;

}

j = 0;

while (\*(s2 + j))

\*(join + i++) = \*(s2 + j++);

\*(join + i) = '\0';

return (join);

}

static void ft\_read\_buffer(int fd, char \*\*holder)

{

char \*buff;

char \*join;

int bytes;

while (1)

{

if (ft\_isline(\*holder) == 1)

break ;

buff = (char \*)malloc(sizeof(char) \* (BUFFER\_SIZE + 1));

if (buff == NULL)

{

\*holder = NULL;

return ;

}

bytes = read(fd, buff, BUFFER\_SIZE);

if (bytes <= 0)

{

free(buff);

return ;

}

\*(buff + bytes) = '\0';

if (\*holder == NULL)

\*holder = buff;

else

{

join = ft\_strjoin(\*holder, buff);

free (buff);

free (holder);

\*holder = join;

}

}

}

**--------------------------------------------------------------**

char \*ft\_extract\_line(char \*\*holder)

{

char \*line;

char \*remainder;

int len;

int i;

int j;

len = ft\_get\_line\_len(\*holder);

if (len == 0)

return (NULL);

line = (char \*)malloc(sizeof(char) \* (len + 1));

if (line == NULL)

return (NULL);

i = 0;

while (i < len)

{

\*(line + i) = \*(\*holder + i);

i++;

}

\*(line + i) = '\0';

remainder = (char \*)malloc(sizeof(char) \* (ft\_strlen(\*holder) + 1));

if (remainder == NULL)

return (NULL);

j= 0;

while (\*(\*holder + i))

{

\*(remainder + j) = \*(\*holder + i);

j++;

i++;

}

\*(remainder + j) = '\0';

free (\*holder);

\*holder = remainder;

return (line);

}

char \*get\_next\_line(int fd)

{

static char \*holder;

char \*line;

if (fd == -1 || fd > FD\_LIMIT || BUFFER\_SIZE < 1)

return (NULL);

ft\_read\_buffer(fd, &holder);

if (holder == NULL || \*holder =='\0')

return (NULL);

line = ft\_extract\_line(&holder);

if (line == NULL)

free(holder);

return (line);

}

**get\_next\_line.h**

#ifndef GET\_NEXT\_LINE\_H

# define GET\_NEXT\_LINE\_H

# include <stdlib.h>

# include <unistd.h>

char \*get\_next\_line(int fd);

#endif