Assignment name : **flood_fill**
Expected files   : *.c, *.h
Allowed functions: -

Write a function that takes a char ** as a 2-dimensional array of char, a t_point as the dimensions of this array and a t_point as the starting point.

Starting from the given 'begin' t_point, this function fills an entire zone by replacing characters inside with the character 'F'. A zone is an group of the same character delimitated horizontally and vertically by other characters or the array boundary.

The flood_fill function won't fill diagonally.

```
#include "flood_fill.h"

void	flood_fill(char **tab, t_point size, t_point begin)
{
	while (begin.y < size.y)
	{
		while (begin.x < size.x)
		{
			if (tab[begin.y][begin.x] == '1')
				tab[begin.y][begin.x] = 'F';
			begin.x++;
		}
		begin.x = 0;
		begin.y++;
	}
}
```

Assignment name : **fprime**
Expected files   : fprime.c
Allowed functions: printf, atoi

Write a program that takes a positive int and displays its prime factors on the standard output, followed by a newline.

Factors must be displayed in ascending order and separated by '*', so that the expression in the output gives the right result.

If the number of parameters is not 1, simply display a newline.
The input, when there is one, will be valid.

```c
#include <unistd.h>

void	ft_putchar(char c);

void	ft_putstr(char *str);

void	ft_putnbr(int n);

int	ft_atoi(char *s);

void	fprime(int nbr)
{
	int	i;

	i = 2;
	if (nbr == 1)
		ft_putchar('1');
	while (nbr >= i)
	{
		if (nbr % i == 0)
		{
			ft_putnbr(i);
			if (nbr == i)
				break ;
			else
				ft_putchar('*');
			nbr /= i;
			i = 2;
		}
		i++;
	}
}

int	main(int argc, char *argv[])
{
	if (argc == 2)
		fprime(ft_atoi(argv[1]));
	ft_putchar('\n');
	return (0);
}
```

Assignment name : **ft_itoa**
Expected files   : ft_itoa.c
Allowed functions: malloc

Write a function that takes an int and converts it to a null-terminated string.
The function returns the result in a char array that you must allocate.

Your function must be declared as follows:
char *ft_itoa(int nbr);

```c
#include <stdlib.h>
#include <stdio.h>

int	nbr_len(int nbr)
{
	int i;

	i = 1;
	if (nbr < 0)
	{
		i++;
		nbr *= -1;
	}
	while (nbr > 9)
	{
		nbr /= 10;
		i++;
	}
	return (i);
}

int	ft_div(int len)
{
	int i;

	i = 1;
	if (len == 1)
		return (1);
	while (len > 1)
	{
		i *= 10;
		len--;
	}
	return (i);
}

int	ft_div(int len)
{
}

char	*ft_itoa(int nbr)
{
	int i;
	int len;
	int len2;
	char *result;

	i = 0;
	len = nbr_len(nbr);
	len2 = len;
	if ((result = (char*)malloc(sizeof(char) * (len + 1))) == NULL)
		return (NULL);
	if (nbr == -2147483648)
		return ("-2147483648\0");
	if (nbr < 0)
	{
		nbr *= -1;
		result[0] = '-';
		i++;
		len--;
	}
	while (i < len2)
	{
		result[i] = ((nbr / ft_div(len)) % 10) + 48;
		len--;
		i++;
	}
	result[i] = '\0';
	return (result);
}

int main(void)
{
```

```c
int main(void)
{
	printf("0 -> %s\n", ft_itoa(0));
	printf("1 -> %s\n", ft_itoa(1));
	printf("42 -> %s\n", ft_itoa(42));
	printf("1001 -> %s\n", ft_itoa(1001));
	printf("0-> %s\n", ft_itoa(-0));
	printf("-2-> %s\n", ft_itoa(-2));
	printf("-24-> %s\n", ft_itoa(-24));
	printf("-2147483648-> %s\n", ft_itoa(-2147483648));
	printf("2147483647-> %s\n", ft_itoa(2147483647));
}
```

Assignment name : **ft_list_foreach**
Expected files   : ft_list_foreach.c, ft_list.h
Allowed functions: -

Write a function that takes a list and a function pointer, and applies this function to each element of the list.

It must be declared as follows:
void   ft_list_foreach(t_list *begin_list, void (*f)(void *));

The function pointed to by f will be used as follows:
(*f)(list_ptr->data);

```c
#include "ft_list.h"

void    ft_list_foreach(t_list *begin_list, void (*f)(void *))
{
    while (begin_list != NULL)
    {
        if (begin_list->data)
            (*f)(begin_list->data);
        begin_list = begin_list->next;
    }
}
```

```c
#include "ft_list.h"

void    ft_putstr(char *str)
{
    while (*str)
        write(1, str++, 1);
}

t_list  *ft_new_elem(void *data)
{
    t_list  *node;

    node = (t_list *)malloc(sizeof(t_list));
    if (!node)
        return (node = NULL);
    node->data = data;
    node->next = NULL;
    return (node);
}
```

```c
t_list  *ft_new_elem(void *data)
}

int main(void)
{
    t_list  *test_list;

    test_list = ft_new_elem("Follow ");
    test_list->next = ft_new_elem("the ");
    test_list->next->next = ft_new_elem("white ");
    test_list->next->next->next = ft_new_elem("rabbit ");
    test_list->next->next->next->next = ft_new_elem(".");
    ft_list_foreach(test_list, (void *)ft_putstr);
    ft_putstr("\n");
    return (0);
}
```

Assignment name : **ft_list_remove_if**
Expected files   : ft_list_remove_if.c
Allowed functions: free

Write a function called ft_list_remove_if that removes from the passed list any element the data of which is "equal" to the reference data.

It will be declared as follows :
void ft_list_remove_if(t_list **begin_list, void *data_ref, int (*cmp)());

cmp takes two void* and returns 0 when both parameters are equal.

You have to use the ft_list.h file, which will contain:
```
typedef struct      s_list
{
    struct s_list   *next;
    void            *data;
}                   t_list;
```

```c
#include "ft_list.h"
#include <stdlib.h>

void	ft_list_remove_if(t_list **begin_list, void *data_ref, int (*cmp)())
{
	t_list *current;
	t_list *last;
	t_list *next;

	current = *begin_list;
	last = ((void *)0);
	while (current)
	{
		next = current->next;
		if (cmp(current->data, data_ref) == 0)
		{
			if (last)
				last->next = current->next;
			else
				*begin_list = current->next;
			free(current);
			current = ((void *)0);
		}
		last = current;
		current = next;
	}
}
```

Assignment name : **ft_split**
Expected files   : ft_split.c
Allowed functions: malloc

Write a function that takes a string, splits it into words, and returns them as a NULL-terminated array of strings.

A "word" is defined as a part of a string delimited either by spaces/tabs/new lines, or by the start/end of the string.

Your function must be declared as follows:
char   **ft_split(char *str);

```c
#include <stdlib.h>
#include <stdio.h>
# define WD_NUM 1000
# define WD_LEN 1000

char    **ft_split(char *str)
{
    int i;
    int j;
    int k;
    char **tab;

    i = 0;
    j = 0;
    tab = (char**)malloc(sizeof(**tab) * WD_NUM);
    while (str[i] == ' ' || str[i] == '\t' || str[i] == '\n')
        i++;
    while (str[i] != '\0')
    {
        if (str[i] > 32)
        {
            k = 0;
            tab[j] = (char*)malloc(sizeof(char) * WD_LEN);
            while (str[i] > 32)
            {
                tab[j][k] = str[i];
                i++;
                k++;
            }
            tab[j][k] = '\0';
            j++;
        }
        else
            i++;
    }
    tab[j] = 0;
    return (tab);
}
```

Assignment name : **rev_wstr**
Expected files   : rev_wstr.c
Allowed functions: write, malloc, free

Write a program that takes a string as a parameter, and prints its words in reverse order.

A "word" is a part of the string bounded by spaces and/or tabs, or the begin/end of the string.

If the number of parameters is different from 1, the program will display '\n'.

In the parameters that are going to be tested, there won't be any "additional" spaces (meaning that there won't be additional spaces at the beginning or at the end of the string, and words will always be separated by exactly one space).

```
void	ft_putchar(char c)
{
	write(1, &c, 1);
}

int is_space(char c)
{
	if ((c == ' ') || (c == '\t'))
		return (1);
	return (0);
}

void	rev_wstr(char *str)
{
	int idx;
	int j;
	int first_word;

	idx = 0;
	first_word = 1;
	while (str[idx] != '\0')
		idx++;
	while (idx >= 0)
	{
		while (idx >= 0 && (str[idx] == '\0' || is_space(str[idx])))
			idx--;
		j = idx;
		while (j >= 0 && !is_space(str[j]))
			j--;
		if (first_word == 0)
			ft_putchar(' ');
		write(1, str + j + 1, idx - j);
		first_word = 0;
		idx = j;
	}
}

int main(int argc, char **argv)
{
	if (argc == 2)
		rev_wstr(argv[1]);
	ft_putchar('\n');
	return (0);
}
```

Assignment name : **rostring**
Expected files   : rostring.c
Allowed functions: write, malloc, free

Write a program that takes a string and displays this string after rotating it
one word to the left.

Thus, the first word becomes the last, and others stay in the same order.

A "word" is defined as a part of a string delimited either by spaces/tabs, or by the
start/end of the string.

Words will be separated by only one space in the output.

If there's less than one argument, the program displays \n.

```c
#include <unistd.h>

void	ft_putchar(char c)
{
	write(1, &c, 1);
}

int is_space(char c)
{
	if ((c == ' ') || (c == '\t'))
		return (1);
	return (0);
}

void	ft_print_first_word(char *str, int begin_space)
{
	while (str[begin_space] != '\0' && !is_space(str[begin_space]))
	{
		ft_putchar(str[begin_space]);
		begin_space++;
	}
}
```

```c
void	rostring(char *str)
{
	int idx;
	int begin_space;

	begin_space = 0;
	while (str[begin_space] != '\0' && is_space(str[begin_space]))
		begin_space++;
	idx = begin_space;
	while (str[idx] != '\0' && !is_space(str[idx]))
		idx++;
	while (str[idx] != '\0')
	{
		if (str[idx] != '\0' && !is_space(str[idx]) && is_space(str[idx - 1]))
		{
			while (str[idx] != '\0' && !is_space(str[idx]))
			{
				ft_putchar(str[idx]);
				idx++;
			}
			ft_putchar(' ');
		}
		idx++;
	}
	ft_print_first_word(str, begin_space);
}

int main(int argc, char **argv)
{
	if (argc > 1)
		rostring(argv[1]);
	ft_putchar('\n');
	return (0);
}
```

Assignment name : **sort_int_tab**
Expected files   : sort_int_tab.c
Allowed functions: –

Write the following function:
void sort_int_tab(int *tab, unsigned int size);

It must sort (in-place) the 'tab' int array, that contains exactly ‘size' members, in ascending order.

Doubles must be preserved. Input is always coherent.

```
void	sort_int_tab(int *tab, unsigned int size)
{
	int			tmp;
	unsigned int	i;

	i = 0;
	while (i < (size - 1))
	{
		if (tab[i] > tab[i + 1])
		{
			tmp = tab[i];
			tab[i] = tab[i + 1];
			tab[i + 1] = tmp;
			i = -1;
		}
		i++;
	}
}
```

Assignment name : **sort_list**
Expected files   : sort_list.c
Allowed functions: –

Write the following functions:
t_list *sort_list(t_list* lst, int (*cmp)(int, int));

This function must sort the list given as a parameter, using the function pointer cmp to select the order to apply, and returns a pointer to the first element of the sorted list.

Duplications must remain. Inputs will always be consistent.

You must use the type t_list described in the file list.h that is provided to you. You must include that file (#include "list.h"), but you must not turn it in. We will use our own to compile your assignment.

Functions passed as cmp will always return a value different from 0 if a and b are in the right order, 0 otherwise.

```c
#include <stdlib.h>
#include "list.h"

void	ft_swap(int *a, int *b)
{
	int tmp;

	tmp = *a;
	*a = *b;
	*b = tmp;
}

t_list	*sort_list(t_list *lst, int (*cmp)(int, int))
{
	t_list	*tmp;

	tmp = lst;
	while (lst->next != NULL)
	{
		if (((*cmp)(lst->data, lst->next->data)) == 0)
		{
			ft_swap(&lst->data, &lst->next->data);
			lst = tmp;
		}
		else
			lst = lst->next;
	}
	lst = tmp;
	return (lst);
}
```