exam01 6 exercices
0-only z  16points
1-rev_print or 1-ft_strlen or 1-ft_swap 16points -> 32/100
2-rot_13 or 2-rotone or 2-first_word or 2-ft_strrev 16points -> 48/100
3-inter or 3-last_word or 3-union or 3-wdmatch 16points -> 64/100
4-ft_range 16points ->  16 points -> 80/100
5-ft_itoa or ft_split ->  20 points -> 100/100

**Assignment name  : only_z**
**Expected files   : only_z.c**
**Allowed functions: write**
**Write a program that displays a 'z' character on the standard output.**

```
#include <unistd.h>

int		main(void)
{
	write(1, "z", 1);
	return (0);
}
```

---

**Assignment name  : ft_strlen**
**Expected files   : ft_strlen.c**
**Allowed functions:**
**Write a function that returns the length of a string.**
**Your function must be declared as follows:**
**int	ft_strlen(char *str);**

```
int		ft_strlen(char *str)
{
	int i;

	i = 0;
	while (str[i])
		i++;
	return (i);
}
```

---

**Assignment name  : ft_swap**
**Expected files   : ft_swap.c**
**Allowed functions:**

**Write a function that swaps the contents of two integers the adresses of which are passed as parameters.**

**Your function must be declared as follows:**

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

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

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

---

**Assignment name : rev_print**
**Expected files : rev_print.c**
**Allowed functions: write**
**Write a function that takes a string and displays the string in reverse order followed by the newline.**
**Its prototype is constructed like this :**
**char *ft_rev_print (char *str)**
**It must return its argument**
**Examples:**

```
$> ./rev_print "zaz" | cat -e
zaz$
$> ./rev_print "dub0 a POIL" | cat -e
LIOP a 0bud$
$> ./rev_print | cat -e
$
```

```
#include <unistd.h>
#include <stdio.h>
int	ft_strlen(char *str)
{
	int	i;
	i = 0;
	while(str[i] != '\0')
		i++;
	return (i);
}
char *ft_rev_print(char *str)
{
	int	i;

	i = ft_strlen(str);
	i--;
	while (i >= 0)
	{
		write(1, &str[i], 1);
		i--;
	}
	return(str);
}

int main()
{
	char str[] = "gbrireugneb";
	ft_rev_print(str);
}
```

---

**Assignment name : first_word**
**Expected files : first_word.c**
**Allowed functions: write**
---

**Write a program that takes a string and displays its first word, followed by a newline.**

**A word is a section of string delimited by spaces/tabs or by the start/end of the string.**

**If the number of parameters is not 1, or if there are no words, simply display a newline.**

**Examples:**

```
$> ./first_word "FOR PONY" | cat -e
FOR$
$> ./first_word "this       ...       is sparta, then again, maybe    not" | cat -e
this$
$> ./first_word "   " | cat -e
$
$> ./first_word "a" "b" | cat -e
$
$> ./first_word "  lorem,ipsum  " | cat -e
lorem,ipsum$
$>
```

```
#include <unistd.h>

int		main(int ac, char **av)
{
	int i;

	i = 0;
	if (ac == 2)
	{
		while (av[1][i] == ' ' || av[1][i] == '\t')
			i++;
		while (av[1][i] != '\0' && av[1][i] != ' ' && av[1][i] != '\t')
		{
			write(1, &av[1][i], 1);
			i++;
		}
	}
	write(1, "\n", 1);
	return (0);
}
```

**Assignment name : ft_strrev**
**Expected files : ft_strrev.c**
**Allowed functions:**
**----------**

**Write a function that reverses (in-place) a string.**

**It must return its parameter.**

**Your function must be declared as follows:**

**char *ft_strrev(char *str);**

```
int             ft_strlen(char *str)
{
        int i;

        i = 0;
        while (str[i] != '\0')
                i++;
        return (i);
}

char    *ft_strrev(char *str)
{
        int i;
        int len;
        char tmp;

        i = 0;
        len = ft_strlen(str) - 1;
        while (len > i)
        {
                tmp = str[i];
                str[i] = str[len];
                str[len] = tmp;
                i++;
                len--;
        }
        return (str);
}
```

**Assignment name : rot_13**
**Expected files : rot_13.c**
**Allowed functions: write**
---

**Write a program that takes a string and displays it, replacing each of its letters by the letter 13 spaces ahead in alphabetical order.**

**'z' becomes 'm' and 'Z' becomes 'M'. Case remains unaffected.**

**The output will be followed by a newline.**

**If the number of arguments is not 1, the program displays a newline.**

**Example:**

```
$>./rot_13 "abc"
nop
$>./rot_13 "My horse is Amazing." | cat -e
Zl ubefr vf Nznmvat.$
$>./rot_13 "AkjhZ zLKIJz , 23y " | cat -e
NxwuM mYXVWm , 23l $
$>./rot_13 | cat -e
$
$>
$>./rot_13 "" | cat -e
$
$>
```

```
#include <unistd.h>
void	rot13(char *str)
{
	int	i;
	i = 0;
	while( str[i] != '\0')
	{
		if( (str[i] >= 'A' && str[i] <= 'M') || (str[i] >= 'a' && str[i] <= 'm') )
			str[i] += 13;
		else if( (str[i] >= 'N' && str[i] <= 'Z') || (str[i] >= 'n' && str[i] <= 'z') )
			str[i] -= 13;
		write(1, &str[i], 1);
		i++;
	}
}

int	main(int ac, char **av)
{
	if (ac == 2)
		rot13(av[1]);
	write(1, "\n", 1);


}
```

**Assignment name : rotone**
**Expected files : rotone.c**
**Allowed functions: write**
---

**Write a program that takes a string and displays it, replacing each of its letters by the next one in alphabetical order.**

**'z' becomes 'a' and 'Z' becomes 'A'. Case remains unaffected.**

**The output will be followed by a \n.**

**If the number of arguments is not 1, the program displays \n.**

**Example:**

```
$>./rotone "abc"
bcd
$>./rotone "Les stagiaires du staff ne sentent pas toujours tres bon." | cat -e
Mft tubhjbjsft ev tubgg of tfoufou qbt upvkpvst usft cpo.$
$>./rotone "AkjhZ zLKIJz , 23y " | cat -e
BlkiA aMLJKa , 23z $
$>./rotone | cat -e
$
$>
$>./rotone "" | cat -e
$
$>
```

```
#include <unistd.h>
void	rotone(char * str)
{
	int i;
	i = 0;
	while(str[i] != '\0')
	{
		if ( (str[i] >= 'A' && str[i] <= 'Y') || (str[i] >= 'a' && str[i] <= 'y'))
			str[i] += 1;
		else if (str[i] == 'Z' || str[i] == 'z')
			str[i] -= 25;
		write (1, &str[i], 1);
		i++;
	}
}

int	main(int ac, char **av)
{
	if (ac == 2)
		rotone(av[1]);
	write(1, "\n", 1);


}
```

**Assignment name : inter**
**Expected files : inter.c**
**Allowed functions: write**
**---**

**Write a program that takes two strings and displays, without doubles, the characters that appear in both strings, in the order they appear in the first one.**

**The display will be followed by a \n.**

**If the number of arguments is not 2, the program displays \n.**

**Examples:**

**$>./inter "padinton" "paqefwtdjetyiytjneytjoeyjnejeyj" | cat -e**
**padinto$**
**$>./inter ddf6vewg64f gtwthgdwthdwfteewhrtag6h4ffdhsd | cat -e**
**df6ewg4$**
**$>./inter "rien" "cette phrase ne cache rien" | cat -e**
**rien$**
**$>./inter | cat -e**
**$**

```
#include <unistd.h>

int		check_doubles(char *str, char c, int pos)
{
	int i;

	i = 0;
	while (i < pos)
	{
		if (str[i] == c)
			return (0);
		i++;
	}
	return (1);
}
void	inter(char *str, char *str1)
{
	int	i;
	int	j;

	i = 0;
	while (str[i] != '\0')
	{
		j = 0;
		while (str1[j] != '\0')
		{
			if (str[i] == str1[j])
			{
				if (check_doubles(str, str[i], i) == 1)
```

```
                                {
                                        write(1, &str[i], 1);
                                        break;
                                }
                        }
                        j++;
                }
                i++;
        }
}
int             main(int ac, char **av)
{

        if (ac == 3)
                inter(av[1], av[2]);
        write(1, "\n", 1);
        return (0);
}
```

---

**Assignment name : last_word**
**Expected files : last_word.c**
**Allowed functions: write**
---

**Write a program that takes a string and displays its last word followed by a \n.**

**A word is a section of string delimited by spaces/tabs or by the start/end of the string.**

**If the number of parameters is not 1, or there are no words, display a newline.**

**Example:**

```
$> ./last_word "FOR PONY" | cat -e
PONY$
$> ./last_word "this       ...      is sparta, then again, maybe    not" | cat -e
not$
$> ./last_word "   " | cat -e
$
$> ./last_word "a" "b" | cat -e
$
$> ./last_word "  lorem,ipsum  " | cat -e
lorem,ipsum$
$>
```

```
#include <unistd.h>
void    last_word(char *str)
{
        int     i = 0;
        while (str[i] != '\0')
                i++;
        i -= 1;
        while(str[i] == '\t' || str[i] == 32)
                i--;
```

```
	while (i > 0)
	{	if(str[i] == 32 && str[i] == '\t')
			break;
		i--;
	}
	i++;
	while (str[i] != '\0' && str[i] != 32 && str[i] != '\t')
	{
		write(1, &str[i], 1);
		i++;
	}
}
int main(int ac, char **av)
{
	if (ac == 2)
		last_word(av[1]);
	write(1, "\n", 1);
}
```

---

**Assignment name : union**
**Expected files : union.c**
**Allowed functions: write**
---

**Write a program that takes two strings and displays, without doubles, the characters that appear in either one of the strings.**

**The display will be in the order characters appear in the command line, and will be followed by a \n.**

**If the number of arguments is not 2, the program displays \n.**

**Example:**

```
$>./union zpadinton "paqefwtdjetyiytjneytjoeyjnejeyj" | cat -e
zpadintoqefwjy$
$>./union ddf6vewg64f gtwthgdwthdwfteewhrtag6h4ffdhsd | cat -e
df6vewg4thras$
$>./union "rien" "cette phrase ne cache rien" | cat -e
rienct phas$
$>./union | cat -e
$
$>
$>./union "rien" | cat -e
$
$>
```

```
#include <unistd.h>

int		check_doubles2(char *str, char c)
{
	int i;

	i = 0;
	while (str[i] != '\0')
	{
		if (str[i] == c)
			return (0);
		i++;
	}
	return (1);
}

int		check_doubles1(char *str, char c, int pos)
{
	int i;

	i = 0;
	while (i < pos)
	{
		if (str[i] == c)
			return (0);
		i++;
	}
	return (1);
}

void	ft_union(char *str, char *str1)
{
	int	i;

	i = 0;
	while (str[i] != '\0')
	{
		if (check_doubles1(str, str[i], i) == 1)
			write(1, &str[i], 1);
		i++;
	}
	i = 0;
	while (str1[i] != '\0')
	{
		if (check_doubles2(str, str1[i]) == 1)
		{
			if (check_doubles1(str1, str1[i], i) == 1)
				write(1, &str1[i], 1);
		}
		i++;
	}
}
```

```
int		main(int ac, char **av)
{
	if (ac == 3)
		ft_union(av[1], av[2]);
	write(1, "\n", 1);
	return (0);
}
```

---

**Assignment name : wdmatch**
**Expected files : wdmatch.c**
**Allowed functions: write**
---

**Write a program that takes two strings and checks whether it's possible to write the first string with characters from the second string, while respecting the order in which these characters appear in the second string.**

**If it's possible, the program displays the string, followed by a \n, otherwise it simply displays a \n.**
**If the number of arguments is not 2, the program displays a \n.**

**Examples:**

```
$>./wdmatch "faya" "fgvvfdxcacpolhyghbreda" | cat -e
faya$
$>./wdmatch "faya" "fgvvfdxcacpolhyghbred" | cat -e
$
$>./wdmatch "quarante deux" "qfqfsudf arzgsayns tsregfdgs sjytdekuoixq " | cat -e
quarante deux$
$>./wdmatch "error" rrerrrfiiljdfxjyuifrrvcoojh | cat -e
$
$>./wdmatch | cat -e
$
```

```
#include <unistd.h>

void	ft_putstr(char *str)
{
	int i;

	i = 0;
	while (str[i] != '\0')
	{
		write(1, &str[i], 1);
		i++;
	}
}

int		ft_strlen(char *str)
{
	int i;

	i = 0;
	while (str[i] != '\0')
```

```
                i++;
        return (i);
}

int             main(int ac, char **av)
{
        int i;
        int j;
        int wdlen;

        i = 0;
        j = 0;
        wdlen = 0;
        if (ac == 3)
        {
                while (av[1][i] != '\0')
                {
                        while (av[2][j] != '\0')
                        {
                                if (av[1][i] == av[2][j])
                                {
                                        wdlen++;
                                        break ;
                                }
                                j++;
                        }
                        i++;
                }
                if (wdlen == ft_strlen(av[1]))
                        ft_putstr(av[1]);
        }
        write(1, "\n", 1);
        return (0);
}
```

---

**Assignment name : ft_range**
**Expected files : ft_range.c**
**Allowed functions: malloc**
**Write the following function:**
**int *ft_range(int start, int end);**
**It must allocate (with malloc()) an array of integers, fill it with consecutive values that begin at start and end at end (Including start and end !), then return a pointer to the first value of the array.**
**Examples:**

**- With (1, 3) you will return an array containing 1, 2 and 3.**
**- With (-1, 2) you will return an array containing -1, 0, 1 and 2.**
**- With (0, 0) you will return an array containing 0.**
**- With (0, -3) you will return an array containing 0, -1, -2 and -3.**

```
#include <stdlib.h>
#include <stdio.h>
int     ft_abs(int x)
{
```

```
    if (x < 0)
        return (-x);
    return (x);
}

int    *ft_range(int start, int end)
{
    int       size;
    int       i;
    int       *tab;
    int       *d;

    size = ft_abs(end - start) + 1;
    d = (tab = malloc(size * sizeof(int)));
    if (!d)
        return (0);
    i = 0;
            if ( size == 1)
            tab[0] = start;
    if (start < end)
    {
        while (i < size)
        {
            tab[i] = start + i;
            i++;
        }
    }
    else if (start > end)
    {
        while (i < size)
        {
            tab[i] = start - i;
            i++;
        }
    }
    return (tab);
}

int       main(void)
{
    int *tab;
    int i = 0;
    int start = 0;
    int end = 0 ;
    int size = ft_abs(end - start) + 1;

    tab = ft_range(start, end);
    while(i < size)
    {
    printf("%i, ", tab[i]);
    i++;
    }
}
```

**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);**

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

int	len(long nb)
{
	int	len = 0;
	if (nb < 0)
	{
		nb *= -1;
		len++;
	}
	while (nb > 0)
	{
		nb /= 10;
		len++;
	}
	return(len);
}
char	*ft_itoa(int nb)
{
	char	*str;
	long	n;
	int	i;

	n = nb;
	i = len(n);
	if(!(str = (char *)malloc(i + 1)))
		return(0);
	str[i--] = '\0';
	if (n == 0)
	{
		str[0] = 48;
		return(str);
	}
	if(n < 0)
	{
		str[0] = '-';
		n *= -1;
	}
	while (n > 0)
	{
		str[i] = 48 + (n % 10);
		n /= 10;
		i--;
	}
```

```
	return (str);
}
int	main(void)
{
	printf("%s\n", ft_itoa(1342345));
}
```

---

**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);**

```
#include <stdlib.h>
int	check_separator(char c)
{
	if ( c == 10 || c == 9 || c == 32)
		return (1);
	if (c == 0)
		return (1);
	return (0);
}
int	count_strings(char *str)
{
	int	i;
	int	count;

	count = 0;
	i = 0;
	while (str[i] != '\0')
	{
		while (str[i] != '\0' && check_separator(str[i]))
			i++;
		if (str[i] != '\0')
			count++;
		while (str[i] != '\0' && !check_separator(str[i]))
			i++;
	}
	return (count);
}
int	ft_strlen_sep(char *str)
{
	int	i;

	i = 0;
	while (str[i] && !check_separator(str[i]))
		i++;
	return (i);
}
char	*ft_word(char *str)
```

```
{
	int		len_word;
	int		i;
	char	*word;

	i = 0;
	len_word = ft_strlen_sep(str);
	word = (char *)malloc(sizeof(char) * (len_word + 1));
	while (i < len_word)
	{
		word[i] = str[i];
		i++;
	}
	word[i] = '\0';
	return (word);
}
char	**ft_split(char *str)
{
	char	**strings;
	int		i;

	i = 0;
	strings = (char **)malloc(sizeof(char *)
			* (count_strings(str) + 1));
	while (*str != '\0')
	{
		while (*str != '\0' && check_separator(*str))
			str++;
		if (*str != '\0')
		{
			strings[i] = ft_word(str);
			i++;
		}
		while (*str && !check_separator(*str))
			str++;
	}
	strings[i] = 0;
	return (strings);
}
#include <stdio.h>
int	main(int argc, char **argv)
{
	int		index;
	char	**split;
	(void)	argc;
	split = ft_split(argv[1]);
	index = 0;
	while (split[index])
	{
		printf("%s\n", split[index]);
		index++;
	}
}
```