Piscine C Exam Review
Piscine C Exam Review Solutions

C Intermediate Exam Review

42 C Beginner Exam Review:

```
=====================================./0-0-aff_a.txt=========================================
Assignment name  : aff_a
Expected files   : aff_a.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program that takes a string, and displays the first 'a' character it
encounters in it, followed by a newline. If there are no 'a' characters in the
string, the program just writes a newline. If the number of parameters is not
1, the program displays 'a' followed by a newline.

Example:

$> ./aff_a "abc" | cat -e
a$
$> ./aff_a "dubO a POIL" | cat -e
a$
$> ./aff_a "zz sent le poney" | cat -e
$
$> ./aff_a | cat -e
a$
=============================================================================================
```

```
=====================================./0-0-ft_countdown.txt=========================================
Assignment name  : ft_countdown
Expected files   : ft_countdown.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program that displays all digits in descending order, followed by a
newline.

Example:
$> ./ft_countdown | cat -e
9876543210$
$>
=============================================================================================
```

```
=====================================./0-0-ft_print_numbers.txt=========================================
Assignment name  : ft_print_numbers
Expected files   : ft_print_numbers.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a function that displays all digits in ascending order.

Your function must be declared as follows:

void    ft_print_numbers(void);

=============================================================================================
```

```
=====================================./0-0-hello.txt=========================================
Assignment name  : hello
Expected files   : hello.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program that displays "Hello World!" followed by a \n.

Example:

$>./hello
Hello World!
$>./hello | cat -e
Hello World!$
$>
=============================================================================================
```

```
=====================================./0-0-maff_alpha.txt=========================================
Assignment name  : maff_alpha
Expected files   : maff_alpha.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program that displays the alphabet, with even letters in uppercase, and
odd letters in lowercase, followed by a newline.
```

```
Example:

$> ./maff_alpha | cat -e
aBcDeFgHiJkLmNoPqRsTuVwXyZ$
================================================================================




======================================./0-1-aff_first_param.txt=========================================
Assignment name  : aff_first_param
Expected files   : aff_first_param.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program that takes strings as arguments, and displays its first
argument followed by a \n.

If the number of arguments is less than 1, the program displays \n.

Example:

$> ./aff_first_param vincent mit "l'ane" dans un pre et "s'en" vint | cat -e
vincent$
$> ./aff_first_param "j'aime le fromage de chevre" | cat -e
j'aime le fromage de chevre$
$> ./aff_first_param
$
================================================================================




======================================./0-1-aff_last_param.txt=========================================
Assignment name  : aff_last_param
Expected files   : aff_last_param.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program that takes strings as arguments, and displays its last
argument followed by a newline.

If the number of arguments is less than 1, the program displays a newline.

Examples:

$> ./aff_last_param "zaz" "mange" "des" "chats" | cat -e
chats$
$> ./aff_last_param "j'aime le savon" | cat -e
j'aime le savon$
$> ./aff_last_param
$
================================================================================




======================================./0-1-maff_revalpha.txt=========================================
Assignment name  : maff_revalpha
Expected files   : maff_revalpha.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program that displays the alphabet in reverse, with even letters in
uppercase, and odd letters in lowercase, followed by a newline.

Example:

$> ./maff_revalpha | cat -e
zYxWvUtSrQpOnMlKjIhGfEdCbA$
================================================================================




======================================./0-1-only_a.txt=========================================
Assignment name  : only_a
Expected files   : only_a.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program that displays a 'a' character on the standard output.
================================================================================




======================================./0-1-only_z.txt=========================================
Assignment name  : only_z
Expected files   : only_z.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program that displays a 'z' character on the standard output.
================================================================================
=====================================./0-2-aff_z.txt=========================================
Assignment name  : aff_z
Expected files   : aff_z.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program that takes a string, and displays the first 'z'
character it encounters in it, followed by a newline. If there are no
'z' characters in the string, the program writes 'z' followed
by a newline. If the number of parameters is not 1, the program displays
'z' followed by a newline.

Example:

$> ./aff_z "abc" | cat -e
z$
$> ./aff_z "dubO a POIL" | cat -e
z$
$> ./aff_z "zaz sent le poney" | cat -e
z$
$> ./aff_z | cat -e
z$
==========================================================================================




=====================================./1-0-ft_strcpy.txt=========================================
Assignment name  : ft_strcpy
Expected files   : ft_strcpy.c
Allowed functions:
--------------------------------------------------------------------------------

Reproduce the behavior of the function strcpy (man strcpy).

Your function must be declared as follows:

char    *ft_strcpy(char *s1, char *s2);
==========================================================================================




=====================================./1-0-ft_strlen.txt=========================================
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);
==========================================================================================




=====================================./1-0-repeat_alpha.txt=========================================
Assignment name  : repeat_alpha
Expected files   : repeat_alpha.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program called repeat_alpha that takes a string and display it
repeating each alphabetical character as many times as its alphabetical index,
followed by a newline.

'a' becomes 'a', 'b' becomes 'bb', 'e' becomes 'eeeee', etc...

Case remains unchanged.

If the number of arguments is not 1, just display a newline.

Examples:

$>./repeat_alpha "abc"
abbccc
$>./repeat_alpha "Alex." | cat -e
Alllllllllllleeeeexxxxxxxxxxxxxxxxxxxxxxxx.$
$>./repeat_alpha 'abacadaba 42!' | cat -e
abbacccaddddabba 42!$
$>./repeat_alpha | cat -e
$
$>
$>./repeat_alpha "" | cat -e
$
$>
==========================================================================================




=====================================./1-0-search_and_replace.txt=========================================
Assignment name  : search_and_replace
```

```
Expected files   : search_and_replace.c
Allowed functions: write, exit
--------------------------------------------------------------------------------

Write a program called search_and_replace that takes 3 arguments, the first
arguments is a string in which to replace a letter (2nd argument) by
another one (3rd argument).

If the number of arguments is not 3, just display a newline.

If the second argument is not contained in the first one (the string)
then the program simply rewrites the string followed by a newline.

Examples:
$>./search_and_replace "Papache est un sabre" "a" "o"
Popoche est un sobre
$>./search_and_replace "zaz" "art" "zul" | cat -e
$
$>./search_and_replace "zaz" "r" "u" | cat -e
zaz$
$>./search_and_replace "jacob" "a" "b" "c" "e" | cat -e
$
$>./search_and_replace "ZoZ eT Dovid oiME le METol." "o" "a" | cat -e
ZaZ eT David aiME le METal.$
$>./search_and_replace "wNcOre Un ExEmPle Pas Facilw a Ecrirw " "w" "e" | cat -e
eNcOre Un ExEmPle Pas Facile a Ecrire $
==========================================================================================




======================================./1-0-ulstr.txt=======================================
Assignment name  : ulstr
Expected files   : ulstr.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program that takes a string and reverses the case of all its letters.
Other characters remain unchanged.

You must display the result followed by a '\n'.

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

Examples :

$>./ulstr "L'eSPrit nE peUt plUs pRogResSer s'Il staGne et sI peRsIsTent VAnIte et auto-justification." | cat -e
l'EspRIT Ne PEuT PLuS PrOGrESsER S'iL STAgNE ET Si PErSiStENT vaNiTE ET AUTO-JUSTIFICATION.$
$>./ulstr "S'enTOuRer dE sECreT eSt uN sIGnE De mAnQuE De coNNaiSSanCe.  " | cat -e
s'ENtoUrER De SecREt EsT Un SigNe dE MaNqUe dE COnnAIssANcE.  $
$>./ulstr "3:21 Ba  tOut  moUn ki Ka di KE m'en Ka fe fot" | cat -e
3:21 bA  ToUT  MOuN KI kA DI ke M'EN kA FE FOT$
$>./ulstr | cat -e
$
==========================================================================================




======================================./1-1-rot_13.txt=======================================
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
$
$>
==========================================================================================




======================================./1-2-first_word.txt=======================================
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
$
$> ./first_word "  lorem,ipsum  " | cat -e
lorem,ipsum$
$>
==========================================================================================




=====================================./1-2-ft_putstr.txt=======================================
Assignment name  : ft_putstr
Expected files   : ft_putstr.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a function that displays a string on the standard output.

The pointer passed to the function contains the address of the string's first
character.

Your function must be declared as follows:

void    ft_putstr(char *str);
==========================================================================================




=====================================./1-2-ft_swap.txt=======================================
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);
==========================================================================================




=====================================./1-3-first_word.txt=======================================
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
$
$> ./first_word "  lorem,ipsum  " | cat -e
lorem,ipsum$
$>
==========================================================================================




=====================================./1-3-rev_print.txt=======================================
Assignment name  : rev_print
Expected files   : rev_print.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program that takes a string, and displays the string in reverse
followed by a newline.

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

Examples:

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




=====================================./1-4-rotone.txt=========================================
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
$
$>
==========================================================================================




=====================================./2-0-ft_atoi.txt=========================================
Assignment name  : ft_atoi
Expected files   : ft_atoi.c
Allowed functions: None
--------------------------------------------------------------------------------

Write a function that converts the string argument str to an integer (type int)
and returns it.

It works much like the standard atoi(const char *str) function, see the man.

Your function must be declared as follows:

int     ft_atoi(const char *str);
==========================================================================================




=====================================./2-0-ft_strdup.txt=========================================
Assignment name  : ft_strdup
Expected files   : ft_strdup.c
Allowed functions: malloc
--------------------------------------------------------------------------------

Reproduce the behavior of the function strdup (man strdup).

Your function must be declared as follows:

char    *ft_strdup(char *src);
==========================================================================================




=====================================./2-0-inter.txt=========================================
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
$
==========================================================================================




======================================./2-0-last_word.txt=========================================
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
$
$> ./last_word "  lorem,ipsum  " | cat -e
lorem,ipsum$
$>
==========================================================================================




======================================./2-0-reverse_bits.txt========================================
Assignment name  : reverse_bits
Expected files   : reverse_bits.c
Allowed functions:
--------------------------------------------------------------------------------

Write a function that takes a byte, reverses it, bit by bit (like the
example) and returns the result.

Your function must be declared as follows:

unsigned char   reverse_bits(unsigned char octet);

Example:

  1 byte
_____________
 0010  0110
         ||
         \/
 0110  0100
==========================================================================================




======================================./2-0-swap_bits.txt=========================================
Assignment name  : swap_bits
Expected files   : swap_bits.c
Allowed functions:
--------------------------------------------------------------------------------

Write a function that takes a byte, swaps its halves (like the example) and
returns the result.

Your function must be declared as follows:

unsigned char   swap_bits(unsigned char octet);

Example:

  1 byte
_____________
 0100 | 0001
```

```
    \ /
    / \
 0001 | 0100
=========================================================================================




======================================./2-0-union.txt=====================================
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
$
$>
=========================================================================================




======================================./2-1-alpha_mirror.txt=====================================
Assignment name  : alpha_mirror
Expected files   : alpha_mirror.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program called alpha_mirror that takes a string and displays this string
after replacing each alphabetical character by the opposite alphabetical
character, followed by a newline.

'a' becomes 'z', 'Z' becomes 'A'
'd' becomes 'w', 'M' becomes 'N'

and so on.

Case is not changed.

If the number of arguments is not 1, display only a newline.

Examples:

$>./alpha_mirror "abc"
zyx
$>./alpha_mirror "My horse is Amazing." | cat -e
Nb slihv rh Znzarmt.$
$>./alpha_mirror | cat -e
$
$>
=========================================================================================




======================================./2-1-max.txt=====================================
Assignment name  : max
Expected files   : max.c
Allowed functions:
--------------------------------------------------------------------------------

Write the following function:

int             max(int* tab, unsigned int len);

The first parameter is an array of int, the second is the number of elements in
the array.

The function returns the largest number found in the array.

If the array is empty, the function returns 0.
=========================================================================================




======================================./2-3-wdmatch.txt=====================================
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
$
==========================================================================================




======================================./2-4-do_op.txt========================================
Assignment name  : do_op
Expected files   : *.c, *.h
Allowed functions: atoi, printf, write
--------------------------------------------------------------------------------

Write a program that takes three strings:
- The first and the third one are representations of base-10 signed integers
  that fit in an int.
- The second one is an arithmetic operator chosen from: + - * / %

The program must display the result of the requested arithmetic operation,
followed by a newline. If the number of parameters is not 3, the program
just displays a newline.

You can assume the string have no mistakes or extraneous characters. Negative
numbers, in input or output, will have one and only one leading '-'. The
result of the operation fits in an int.

Examples:

$> ./do_op "123" "*" 456 | cat -e
56088$
$> ./do_op "9828" "/" 234 | cat -e
42$
$> ./do_op "1" "+" "-43" | cat -e
-42$
$> ./do_op | cat -e
$
==========================================================================================




======================================./2-4-print_bits.txt========================================
Assignment name  : print_bits
Expected files   : print_bits.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a function that takes a byte, and prints it in binary WITHOUT A NEWLINE
AT THE END.

Your function must be declared as follows:

void    print_bits(unsigned char octet);

Example, if you pass 2 to print_bits, it will print "00000010"
==========================================================================================




======================================./2-5-ft_strcmp.txt========================================
Assignment name  : ft_strcmp
Expected files   : ft_strcmp.c
Allowed functions:
--------------------------------------------------------------------------------

Reproduce the behavior of the function strcmp (man strcmp).

Your function must be declared as follows:

int    ft_strcmp(char *s1, char *s2);
==========================================================================================
=====================================./2-5-ft_strrev.txt========================================
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);
==========================================================================================
```

```
=====================================./2-6-is_power_of_2.txt========================================
Assignment name  : is_power_of_2
Expected files   : is_power_of_2.c
Allowed functions: None
--------------------------------------------------------------------------------

Write a function that determines if a given number is a power of 2.

This function returns 1 if the given number is a power of 2, otherwise it returns 0.

Your function must be declared as follows:

int         is_power_of_2(unsigned int n);
==========================================================================================
```

```
=====================================./3-0-add_prime_sum.txt========================================
Assignment name  : add_prime_sum
Expected files   : add_prime_sum.c
Allowed functions: write, exit
--------------------------------------------------------------------------------

Write a program that takes a positive integer as argument and displays the sum
of all prime numbers inferior or equal to it followed by a newline.

If the number of arguments is not 1, or the argument is not a positive number,
just display 0 followed by a newline.

Yes, the examples are right.

Examples:

$>./add_prime_sum 5
10
$>./add_prime_sum 7 | cat -e
17$
$>./add_prime_sum | cat -e
0$
$>
==========================================================================================
```

```
=====================================./3-0-epur_str.txt========================================
Assignment name  : epur_str
Expected files   : epur_str.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program that takes a string, and displays this string with exactly one
space between words, with no spaces or tabs either at the beginning or the end,
followed by a \n.

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

If the number of arguments is not 1, or if there are no words to display, the
program displays \n.

Example:

$> ./epur_str "vous voyez c'est facile d'afficher la meme chose" | cat -e
vous voyez c'est facile d'afficher la meme chose$
$> ./epur_str " seulement          la c'est      plus dur " | cat -e
seulement la c'est plus dur$
$> ./epur_str "comme c'est cocasse" "vous avez entendu, Mathilde ?" | cat -e
$
$> ./epur_str "" | cat -e
$
$>
==========================================================================================
```

```
=====================================./3-0-ft_list_size.txt========================================
Assignment name  : ft_list_size
```

```
Expected files   : ft_list_size.c, ft_list.h
Allowed functions:
--------------------------------------------------------------------------------

Write a function that returns the number of elements in the linked list that's
passed to it.

It must be declared as follows:

int     ft_list_size(t_list *begin_list);

You must use the following structure, and turn it in as a file called
ft_list.h:

typedef struct    s_list
{
    struct s_list *next;
    void          *data;
}                 t_list;
==========================================================================================




======================================./3-0-ft_rrange.txt========================================
Assignment name  : ft_rrange
Expected files   : ft_rrange.c
Allowed functions: malloc
--------------------------------------------------------------------------------

Write the following function:

int     *ft_rrange(int start, int end);

It must allocate (with malloc()) an array of integers, fill it with consecutive
values that begin at end and end at start (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 3, 2 and 1
- With (-1, 2) you will return an array containing 2, 1, 0 and -1.
- With (0, 0) you will return an array containing 0.
- With (0, -3) you will return an array containing -3, -2, -1 and 0.
==========================================================================================




======================================./3-0-hidenp.txt========================================
Assignment name  : hidenp
Expected files   : hidenp.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program named hidenp that takes two strings and displays 1
followed by a newline if the first string is hidden in the second one,
otherwise displays 0 followed by a newline.

Let s1 and s2 be strings. We say that s1 is hidden in s2 if it's possible to
find each character from s1 in s2, in the same order as they appear in s1.
Also, the empty string is hidden in any string.

If the number of parameters is not 2, the program displays a newline.

Examples :

$>./hidenp "fgex.;" "tyf34gdgf;'ektufjhgdgex.;.;rtjynur6" | cat -e
1$
$>./hidenp "abc" "2altrb53c.sse" | cat -e
1$
$>./hidenp "abc" "btarc" | cat -e
0$
$>./hidenp | cat -e
$
$>
==========================================================================================




======================================./3-0-pgcd.txt========================================
Assignment name  : pgcd
Expected files   : pgcd.c
Allowed functions: printf, atoi, malloc, free
--------------------------------------------------------------------------------

Write a program that takes two strings representing two strictly positive
integers that fit in an int.

Display their highest common denominator followed by a newline (It's always a
strictly positive integer).

If the number of parameters is not 2, display a newline.

Examples:

$> ./pgcd 42 10 | cat -e
```

```
2$
$> ./pgcd 42 12 | cat -e
6$
$> ./pgcd 14 77 | cat -e
7$
$> ./pgcd 17 3 | cat -e
1$
$> ./pgcd | cat -e
$
================================================================================




=====================================./3-0-print_hex.txt========================================
Assignment name  : print_hex
Expected files   : print_hex.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program that takes a positive (or zero) number expressed in base 10,
and displays it in base 16 (lowercase letters) followed by a newline.

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

Examples:

$> ./print_hex "10" | cat -e
a$
$> ./print_hex "255" | cat -e
ff$
$> ./print_hex "5156454" | cat -e
4eae66$
$> ./print_hex | cat -e
$
================================================================================




=====================================./3-0-rstr_capitalizer.txt========================================
Assignment name  : rstr_capitalizer
Expected files   : rstr_capitalizer.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program that takes one or more strings and, for each argument, puts
the last character of each word (if it's a letter) in uppercase and the rest
in lowercase, then displays the result followed by a \n.

A word is a section of string delimited by spaces/tabs or the start/end of the
string. If a word has a single letter, it must be capitalized.

If there are no parameters, display \n.

Examples:

$> ./rstr_capitalizer | cat -e
$
$> ./rstr_capitalizer "Premier PETIT TesT" | cat -e
premieR petiT tesT$
$> ./rstr_capitalizer "DeuxiEmE tEST uN PEU moinS  facile" "   attention C'EST pas dur QUAND mEmE" "ALLer UN DeRNier 0123456789pour LA
deuxiemE tesT uN peU moinS  facilE$
   attentioN c'esT paS duR quanD memE$
alleR uN dernieR 0123456789pouR lA routE    E $
$>
================================================================================




=====================================./3-1-expand_str.txt========================================
Assignment name  : expand_str
Expected files   : expand_str.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program that takes a string and displays it with exactly three spaces
between each word, with no spaces or tabs either at the beginning or the end,
followed by a newline.

A word is a section of string delimited either 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:

$> ./expand_str "vous   voyez   c'est   facile   d'afficher   la   meme   chose" | cat -e
vous   voyez   c'est   facile   d'afficher   la   meme   chose$
$> ./expand_str " seulement          la c'est      plus dur " | cat -e
seulement   la   c'est   plus   dur$
$> ./expand_str "comme c'est cocasse" "vous avez entendu, Mathilde ?" | cat -e
$
$> ./expand_str "" | cat -e
$
$>
```

```
==========================================================================================




=====================================./3-1-tab_mult.txt=======================================
Assignment name  : tab_mult
Expected files   : tab_mult.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program that displays a number's multiplication table.

The parameter will always be a strictly positive number that fits in an int,
and said number times 9 will also fit in an int.

If there are no parameters, the program displays \n.

Examples:

$>./tab_mult 9
1 x 9 = 9
2 x 9 = 18
3 x 9 = 27
4 x 9 = 36
5 x 9 = 45
6 x 9 = 54
7 x 9 = 63
8 x 9 = 72
9 x 9 = 81
$>./tab_mult 19
1 x 19 = 19
2 x 19 = 38
3 x 19 = 57
4 x 19 = 76
5 x 19 = 95
6 x 19 = 114
7 x 19 = 133
8 x 19 = 152
9 x 19 = 171
$>
$>./tab_mult | cat -e
$
$>
==========================================================================================




=====================================./3-2-ft_atoi_base.txt=======================================
Assignment name  : ft_atoi_base
Expected files   : ft_atoi_base.c
Allowed functions: None
--------------------------------------------------------------------------------

Write a function that converts the string argument str (base N <= 16)
to an integer (base 10) and returns it.

The characters recognized in the input are: 0123456789abcdef
Those are, of course, to be trimmed according to the requested base. For
example, base 4 recognizes "0123" and base 16 recognizes "0123456789abcdef".

Uppercase letters must also be recognized: "12fdb3" is the same as "12FDB3".

Minus signs ('-') are interpreted only if they are the first character of the
string.

Your function must be declared as follows:

int     ft_atoi_base(const char *str, int str_base);
==========================================================================================




=====================================./3-3-ft_range.txt=======================================
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.
==========================================================================================
=====================================./3-4-paramsum.txt=========================================
Assignment name  : paramsum
Expected files   : paramsum.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program that displays the number of arguments passed to it, followed by
a newline.

If there are no arguments, just display a 0 followed by a newline.

Example:

$>./paramsum 1 2 3 5 7 24
6
$>./paramsum 6 12 24 | cat -e
3$
$>./paramsum | cat -e
0$
$>
==========================================================================================




=====================================./3-4-str_capitalizer.txt=========================================
Assignment name  : str_capitalizer
Expected files   : str_capitalizer.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a program that takes one or several strings and, for each argument,
capitalizes the first character of each word (If it's a letter, obviously),
puts the rest in lowercase, and displays the result on the standard output,
followed by a \n.

A "word" is defined as a part of a string delimited either by spaces/tabs, or
by the start/end of the string. If a word only has one letter, it must be
capitalized.

If there are no arguments, the progam must display \n.

Example:

$> ./str_capitalizer | cat -e
$
$> ./str_capitalizer "Premier PETIT TesT" | cat -e
Premier Petit Test$
$> ./str_capitalizer "DeuxiEmE tEST uN PEU moinS  facile" "   attention C'EST pas dur QUAND mEmE" "ALLer UN DeRNier 0123456789pour LA
Deuxieme Test Un Peu Moins  Facile$
   Attention C'est Pas Dur Quand Meme$
Aller Un Dernier 0123456789pour La Route    E $
$>
==========================================================================================




=====================================./3-4-lcm.txt=========================================
Assignment name  : lcm
Expected files   : lcm.c
Allowed functions:
--------------------------------------------------------------------------------

Write a function who takes two unsigned int as parameters and returns the
computed LCM of those parameters.

LCM (Lowest Common Multiple) of two non-zero integers is the smallest postive
integer divisible by the both integers.

A LCM can be calculated in two ways:

- You can calculate every multiples of each integers until you have a common
multiple other than 0

- You can use the HCF (Highest Common Factor) of these two integers and
calculate as follows:

        LCM(x, y) = | x * y | / HCF(x, y)

  | x * y | means "Absolute value of the product of x by y"

If at least one integer is null, LCM is equal to 0.

Your function must be prototyped as follows:

  unsigned int    lcm(unsigned int a, unsigned int b);
==========================================================================================




=====================================./4-0-fprime.txt=========================================
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's one, will be valid.

Examples:

$> ./fprime 225225 | cat -e
3*3*5*5*7*11*13$
$> ./fprime 8333325 | cat -e
3*3*5*5*7*11*13*37$
$> ./fprime 9539 | cat -e
9539$
$> ./fprime 804577 | cat -e
804577$
$> ./fprime 42 | cat -e
2*3*7$
$> ./fprime 1 | cat -e
1$
$> ./fprime | cat -e
$
$> ./fprime 42 21 | cat -e
$
==========================================================================================




=====================================./4-0-ft_list_foreach.txt=========================================
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);

You must use the following structure, and turn it in as a file called
ft_list.h:

typedef struct    s_list
{
    struct s_list *next;
    void          *data;
}                 t_list;
==========================================================================================




=====================================./4-0-ft_split.txt=========================================
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);
==========================================================================================




=====================================./4-1-rev_wstr.txt=========================================
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 additionnal spaces at the beginning or at
the end of the string, and words will always be separated by exactly one space).

Examples:

$> ./rev_wstr "le temps du mepris precede celui de l'indifference" | cat -e
l'indifference de celui precede mepris du temps le$
$> ./rev_wstr "abcdefghijklm"
abcdefghijklm
$> ./rev_wstr "il contempla le mont" | cat -e
mont le contempla il$
$> ./rev_wstr | cat -e
$
$>
================================================================================




=====================================./4-2-ft_list_remove_if.txt========================================
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:

$>cat ft_list.h
typedef struct      s_list
{
    struct s_list   *next;
    void            *data;
}                   t_list;
$>
================================================================================




=====================================./4-3-sort_int_tab.txt========================================
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.
================================================================================




=====================================./4-3-sort_list.txt========================================
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.
```

```
For example, the following function used as cmp will sort the list
in ascending order:

int ascending(int a, int b)
{
        return (a <= b);
}
==========================================================================================




======================================./4-4-ft_itoa.txt=========================================
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);
==========================================================================================




======================================./4-5-check_mate.txt=========================================
Assignment name  : checkmate
Expected files   : *.c, *.h
Allowed functions: write, malloc, free
--------------------------------------------------------------------------------

Write a program who takes rows of a chessboard in argument and check if your
King is in a check position.

Chess is played on a chessboard, a squared board of 8-squares length with
specific pieces on it : King, Queen, Bishop, Knight, Rook and Pawns.
For this exercice, you will only play with Pawns, Bishops, Rooks and Queen...
and obviously a King.

Each piece have a specific method of movement, and all patterns of capture are
detailled in the examples.txt file.

A piece can capture only the first ennemy piece it founds on its capture
patterns.

The board have a variable size but will remains a square. There's only one King
and all other pieces are against it. All other characters except those used for
pieces are considered as empty squares.

The King is considered as in a check position when an other enemy piece can
capture it. When it's the case, you will print "Success" on the standard output
followed by a newline, otherwise you will print "Fail" followed by a newline.

If there is no arguments, the program will only print a newline.

Examples:

$> ./check_mate '..' '.K' | cat -e
Fail$
$> ./check_mate 'R...' '.K..' '..P.' '....' | cat -e
Success$
$> ./check_mate 'R...' 'iheK' '....' 'jeiR' | cat -e
Success$
$> ./check_mate | cat -e
$
$>
*************************************************
Some subject.en.txts on the web have this example:
$> ./chessmate 'R...' '..P.' '.K..' '....' | cat -e
Success$
Which would indicate that checks need to be down both ways.
Most solutions will:
Fail$
As they are only checking in one direction.
*************************************************
==========================================================================================




======================================./4-6-rostring.txt=========================================
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.

Example:

$>./rostring "abc   " | cat -e
abc$
$>
$>./rostring "Que la      lumiere soit et la lumiere fut"
la lumiere soit et la lumiere fut Que
$>
$>./rostring "     AkjhZ zLKIJz , 23y"
zLKIJz , 23y AkjhZ
$>
$>./rostring | cat -e
$
$>
==========================================================================================




======================================./5-0-brainfuck.txt=========================================
Assignment name  : brainfuck
Expected files   : *.c, *.h
Allowed functions: write, malloc, free
--------------------------------------------------------------------------------

Write a Brainfuck interpreter program.
The source code will be given as first parameter.
The code will always be valid, with no more than 4096 operations.
Brainfuck is a minimalist language. It consists of an array of bytes
(in our case, let's say 2048 bytes) initialized to zero,
and a pointer to its first byte.

Every operator consists of a single character :
- '>' increment the pointer ;
- '<' decrement the pointer ;
- '+' increment the pointed byte ;
- '-' decrement the pointed byte ;
- '.' print the pointed byte on standard output ;
- '[' go to the matching ']' if the pointed byte is 0 (while start) ;
- ']' go to the matching '[' if the pointed byte is not 0 (while end).

Any other character is a comment.

Examples:

$>./brainfuck "++++++++++[>+++++++>++++++++++>+++>+<<<<-] >++.>+.+++++++..+++.>++.<<+++++++++++++++.>.+++.------.--------.>+.>." | cat
Hello World!$
$>./brainfuck "+++++[>++++[>++++H>+++++i<<-]>>>++\n<<<<-]>>--------.>+++++.>." | cat -e
Hi$
$>./brainfuck | cat -e
$
==========================================================================================




======================================./5-1-print_memory.txt=========================================
Assignment name  : print_memory
Expected files   : print_memory.c
Allowed functions: write
--------------------------------------------------------------------------------

Write a function that takes (const void *addr, size_t size), and displays the
memory as in the example.

Your function must be declared as follows:

void    print_memory(const void *addr, size_t size);

---------
$> cat main.c
void    print_memory(const void *addr, size_t size);

int     main(void)
{
        int     tab[10] = {0, 23, 150, 255,
                      12, 16,  21, 42};

        print_memory(tab, sizeof(tab));
        return (0);
}
$> gcc -Wall -Wall -Werror main.c print_memory.c && ./a.out | cat -e
0000 0000 1700 0000 9600 0000 ff00 0000 ................$
0c00 0000 1000 0000 1500 0000 2a00 0000 ............*...$
0000 0000 0000 0000                     ........$

==========================================================================================




======================================./5-2-ft_itoa_base.txt=========================================
Assignment name  : ft_itoa_base
Expected files   : ft_itoa_base.c
```

```
Allowed functions: malloc
--------------------------------------------------------------------------------

Write a function that converts an integer value to a null-terminated string
using the specified base and stores the result in a char array that you must
allocate.

The base is expressed as an integer, from 2 to 16. The characters comprising
the base are the digits from 0 to 9, followed by uppercase letter from A to F.

For example, base 4 would be "0123" and base 16 "0123456789ABCDEF".

If base is 10 and value is negative, the resulting string is preceded with a
minus sign (-). With any other base, value is always considered unsigned.

Your function must be declared as follows:

char	*ft_itoa_base(int value, int base);
==========================================================================================




=====================================./5-3-brackets.txt=========================================
Assignment name  : brackets
Expected files   : *.c *.h
Allowed functions: write
--------------------------------------------------------------------------------

Write a program that takes an undefined number of strings in arguments. For each
argument, the program prints on the standard output "OK" followed by a newline
if the expression is correctly bracketed, otherwise it prints "Error" followed by
a newline.

Symbols considered as 'brackets' are brackets '(' and ')', square brackets '['
and ']'and braces '{' and '}'. Every other symbols are simply ignored.

An opening bracket must always be closed by the good closing bracket in the
correct order. A string which not contains any bracket is considered as a
correctly bracketed string.

If there is no arguments, the program must print only a newline.

Examples :

$> ./brackets '(johndoe)' | cat -e
OK$
$> ./brackets '([)]' | cat -e
Error$
$> ./brackets '' '{[(0 + 0)(1 + 1)](3*(-1)){()}}' | cat -e
OK$
OK$
$> ./brackets | cat -e
$
$>
==========================================================================================




=====================================./5-4-rpn_calc.txt=========================================
Assignment name  : rpn_calc
Expected files   : *.c, *.h
Allowed functions: atoi, printf, write, malloc, free
--------------------------------------------------------------------------------

Write a program that takes a string which contains an equation written in
Reverse Polish notation (RPN) as its first argument, evaluates the equation, and
prints the result on the standard output followed by a newline.

Reverse Polish Notation is a mathematical notation in which every operator
follows all of its operands. In RPN, every operator encountered evaluates the
previous 2 operands, and the result of this operation then becomes the first of
the two operands for the subsequent operator. Operands and operators must be
spaced by at least one space.

You must implement the following operators : "+", "-", "*", "/", and "%".

If the string isn't valid or there isn't exactly one argument, you must print
"Error" on the standard output followed by a newline.

All the given operands must fit in a "int".

Examples of formulas converted in RPN:

3 + 4                   >>    3 4 +
((1 * 2) * 3) - 4       >>    1 2 * 3 * 4 -  ou  3 1 2 * * 4 -
50 * (5 - (10 / 9))     >>    5 10 9 / - 50 *

Here's how to evaluate a formula in RPN:

1 2 * 3 * 4 -
2 3 * 4 -
6 4 -
2

Or:
```

```
3 1 2 * * 4 -
3 2 * 4 -
6 4 -
2

Examples:

$> ./rpn_calc "1 2 * 3 * 4 +" | cat -e
10$
$> ./rpn_calc "1 2 3 4 +" | cat -e
Error$
$> ./rpn_calc |cat -e
Error$
==========================================================================================




======================================./5-5-options.txt========================================
Assignment name  : options
Expected files   : *.c *.h
Allowed functions: write
--------------------------------------------------------------------------------

Write a program that takes an undefined number of arguments which could be
considered as options and writes on standard output a representation of those
options as groups of bytes followed by a newline.

An option is an argument that begins by a '-' and have multiple characters
which could be : abcdefghijklmnopqrstuvwxyz

All options are stocked in a single int and each options represents a bit of that
int, and should be stocked like this :

00000000 00000000 00000000 00000000
******zy xwvutsrq ponmlkji hgfedcba

Launch the program without arguments or with the '-h' flag activated must print
an usage on the standard output, as shown in the following examples.

A wrong option must print "Invalid Option" followd by a newline.

Examples :
$>./options
options: abcdefghijklmnopqrstuvwxyz
$>./options -abc -ijk
00000000 00000000 00000111 00000111
$>./options -z
00000010 00000000 00000000 00000000
$>./options -abc -hijk
options: abcdefghijklmnopqrstuvwxyz
$>./options -%
Invalid Option
==========================================================================================
```

Piscine C Exam Review
Piscine C Exam Review Solutions

C Intermediate Exam Review