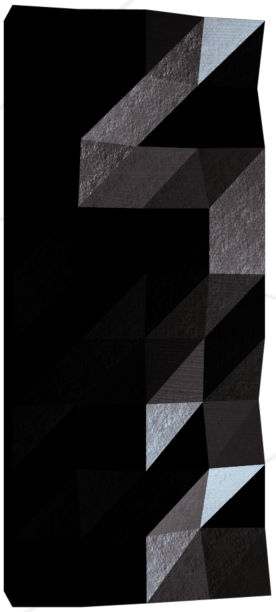

Libft

Your very first own library

Summary: 본 프로젝트의 목적은 흔히 쓰이는 함수들을 재편성한 C 라이브러리를 코드화하기 위한 것입니다. 그것은 다른 모든 프로젝트에서 사용할 수 있게 될 것입니다.

Contents

Chapter I

Introduction

매우 유용한 표준 함수들을 사용할 수 없을 때 C 프로그래밍이 매우 지루해질 수 있습니다. 본 프로젝트는 이런 함수들을 다시 쓰고, 그것들을 이해하고, 그 함수들의 사용법을 배울 기회를 제공합니다. 이 라이브러리는 나중에 진행할 C 프로젝트에 도움이 될 것입니다.

본 프로젝트를 통해서, 우리는 네가 만든 함수 리스트를 확장할 수 있는 기회를 제공합니다. 시간을 내서 일 년 내내 너의 libft를 확장하세요.

Chapter II

Common Instructions

- 프로젝트는 Norm 규칙에 맞춰 작성되어야 합니다. 보너스 파일/함수가 있는 경우, 해당 파일/함수들은 norm 검사에 포함되며, norm error가 있을 시, 0점을 받게 될것입니다.
- 함수들은 정의되지 않은 행동들과는 별개로 예기치 않게 중단되어서는 안 됩니다. (예를 들어, segmentation fault, bus error, double free 등.)
- 필요한 경우 heap에 할당된 모든 메모리 공간은 적절하게 해제되어야 합니다. 메모리 누수는 용납되지 않을 것입니다.
- 그 과제가 필요하다면, Makefile을 제출해야 합니다. 그것은 -Wall -Wextra -Werror 플래그를 지정하여 컴파일할 것입니다. 그리고 Makefile은 relink 되어서는 안됩니다.
- Makefile은 최소한 $(NAME), all, clean, fclean, re를 포함해야 합니다.
- 프로젝트에 보너스를 제출하려면, Makefile에 보너스 규칙을 포함해야 합니다. 이보너스 규칙은 프로젝트의 메인 부분에서 금지되었던 모든 다양한 헤더, 라이브러리, 또는 함수들은 추가해야 할 것입니다. 보너스는 반드시 _bonus.{c/h}라는 다른 파일에 있어야 합니다. 의무적으로 해야 될 파트과 보너스 파트는 별도로 평가될 것입니다.
- 프로젝트에서 당신의 libft를 허용한다면, 소스들과 그것과 연관된 Makefile을 연관된 Makefile과 함께 libft폴더에 복사해야 합니다. 프로젝트의 Makefile은 Makefile을 사용하여 라이브러리를 컴파일한 다음, 프로젝트를 컴파일해야 합니다.
- 이 과제물을 제출하지 않고 등급이 매겨지지 않을지라도, 우리는 당신의 프로젝트를 위한 테스트 프로그램을 만들 것을 권장합니다. 그것은 너의 work와 peer's work를 쉽게 테스트할 기회를 제공할 것입니다. 너는 defence하는 동안 이 테스트 프로그램들이 특히 유용하다는 것을 알게 될 것입니다. 사실, defence하는 동안, 너는 너의 테스트 프로그램과 평가 받는 동료의 테스트 프로그램들을 자유롭게 사용할 수 있을것입니다.
- 할당된 git 저장소에 과제물을 제출하세요. 오직 git 저장소에 있는 과제물만 등급이 매겨질 것입니다. 만약 너의 과제를 평가받는데 Deepthought가 배정된다면, 그것은 동료평가 이후에 이루어질 것입니다. 만약 Deepthought 평가 중에 오류가 발생한다면, 그 즉시 평가는 중지될 것입니다.

Chapter III

Mandatory part

Program name	`libft.a`
Turn in files	`*.c, libft.h, Makefile`
Makefile	`Yes`
External functs.	`Detailed below`
Libft authorized	`Non-applicable`
Description	`Write your own library, containing an extract of important functions for your cursus.`

III.1 Technical considerations

- It is forbidden to use global variables.
- If you need subfunctions to write a complex function, you should define these subfunctions as `static` to avoid publishing them with your library. It would be a good habit to do this in your future projects as well.
- Submit all files in the root of your repository.

- 전역 변수는 사용할 수 없습니다.
- 복잡한 함수를 작성하기 위해 하위 함수가 필요한 경우에는, 이러한 하위 함수를 라이브러리와 함께 publishing 하지 않도록 static(정적)으로 정의해야 합니다.
- 저장소의 root에 있는 모든 파일을 제출하세요.

III.2 Part 1 - Libc functions

첫 번째 파트에서는, man에 정의되어 있는 대로 libc functions의 set을 다시 코드화해야 합니다. 함수들은 원본과 같은 형식의 프로토타입을 선언해야 할 것입니다.
함수의 이름 앞에는 “ft_” 를 붙여야 합니다. 예를 들어 strlen은 다음과 같이 됩니다. **ft_strlen.**

re-code해야하는 함수의 프로토타입의 일부는 “restrict” 한정자를 사용합니다. 이 키워드는 c99 표준의 일부분입니다.
그러므로 프로토타입에 포함시키고 -std=c99 플래그를 사용하여 컴파일 하는 것은 금지됩니다.

아래의 함수들을 다시 코드화해야 합니다. 이 함수들은 외부 함수들을 필요로 않습니다.

- `memset`
- `bzero`
- `memcpy`
- `memccpy`
- `memmove`
- `memchr`
- `memcmp`
- `strlen`
- `strlcpy`
- `strlcat`
- `strchr`
- `strrchr`
- `strnstr`
- `strncmp`
- `atoi`
- `isalpha`
- `isdigit`
- `isalnum`
- `isascii`
- `isprint`
- `toupper`
- `tolower`

아래의 함수들 또한 “malloc” 함수를 사용하여, 다시 코드화 해야 합니다.

- `calloc`
- `strdup`

III.3 Part 2 - Additional functions

두 번째 파트에서는, libc에 포함되있지 않거나 다른 형식으로 포함된 functions의 set을 코드화해야 합니다. 이러한 함수 중 일부는 파트1의 함수들을 쓰는 데 유용할 수 있습니다.

Function name	ft_substr
Prototype	char *ft_substr(char const *s, unsigned int start, size_t len);
Turn in files	-
Parameters	#1. The string from which to create the substring. #2. The start index of the substring in the string 's'. #3. The maximum length of the substring.
Return value	The substring. NULL if the allocation fails.
External functs.	malloc
Description	Allocates (with malloc(3)) and returns a substring from the string 's'. The substring begins at index 'start' and is of maximum size 'len'.

Function name	ft_strjoin
Prototype	char *ft_strjoin(char const *s1, char const *s2);
Turn in files	-
Parameters	#1. The prefix string. #2. The suffix string.
Return value	The new string. NULL if the allocation fails.
External functs.	malloc
Description	Allocates (with malloc(3)) and returns a new string, which is the result of the concatenation of 's1' and 's2'.

Function name	ft_strtrim
Prototype	char *ft_strtrim(char const *s1, char const *set);
Turn in files	-
Parameters	#1. The string to be trimmed. #2. The reference set of characters to trim.
Return value	The trimmed string. NULL if the allocation fails.
External functs.	malloc
Description	Allocates (with malloc(3)) and returns a copy of 's1' with the characters specified in 'set' removed from the beginning and the end of the string.

Function name	ft_split
Prototype	char **ft_split(char const *s, char c);
Turn in files	-
Parameters	#1. The string to be split. #2. The delimiter character.
Return value	The array of new strings resulting from the split. NULL if the allocation fails.
External functs.	malloc, free
Description	Allocates (with malloc(3)) and returns an array of strings obtained by splitting 's' using the character 'c' as a delimiter. The array must be ended by a NULL pointer.

Function name	ft_itoa
Prototype	char *ft_itoa(int n);
Turn in files	-
Parameters	#1. the integer to convert.
Return value	The string representing the integer. NULL if the allocation fails.
External functs.	malloc
Description	Allocates (with malloc(3)) and returns a string representing the integer received as an argument. Negative numbers must be handled.

Function name	ft_strmapi
Prototype	char *ft_strmapi(char const *s, char (*f)(unsigned int, char));
Turn in files	-
Parameters	#1. The string on which to iterate. #2. The function to apply to each character.
Return value	The string created from the successive applications of 'f'. Returns NULL if the allocation fails.
External functs.	malloc
Description	Applies the function 'f' to each character of the string 's' to create a new string (with malloc(3)) resulting from successive applications of 'f'.

Function name	`ft_putchar_fd`
Prototype	`void ft_putchar_fd(char c, int fd);`
Turn in files	-
Parameters	#1. The character to output. #2. The file descriptor on which to write.
Return value	None
External functs.	`write`
Description	Outputs the character 'c' to the given file descriptor.

Function name	`ft_putstr_fd`
Prototype	`void ft_putstr_fd(char *s, int fd);`
Turn in files	-
Parameters	#1. The string to output. #2. The file descriptor on which to write.
Return value	None
External functs.	`write`
Description	Outputs the string 's' to the given file descriptor.

Function name	`ft_putendl_fd`
Prototype	`void ft_putendl_fd(char *s, int fd);`
Turn in files	-
Parameters	#1. The string to output. #2. The file descriptor on which to write.
Return value	None
External functs.	`write`
Description	Outputs the string 's' to the given file descriptor, followed by a newline.

Function name	`ft_putnbr_fd`
Prototype	`void ft_putnbr_fd(int n, int fd);`
Turn in files	-
Parameters	#1. The integer to output. #2. The file descriptor on which to write.
Return value	None
External functs.	`write`
Description	Outputs the integer 'n' to the given file descriptor.

Chapter IV

Bonus part

필수로 수행해야 하는 부분을 성공적으로 완료했다면, 너는 더 나아가 즐길 수 있을 것입니다. 이 마지막 섹션을 보너스 점수로 볼 수 있습니다.

메모리와 문자열을 조작하는 함수를 갖는 것은 매우 유용하지만, 곧 리스트를 조작하는 함수를 갖는 것이 훨씬 더 유용하다는 것을 알게 될 것입니다.

다음의 구조를 사용하여 리스트의 요소들을 표현하세요. 이 구조를 libft.h 파일에 추가해야 합니다.

보너스를 만들면 libft.a 라이브러리에 보너스 함수들이 추가될 것입니다.

이 파트의 헤더와 .c파일에 보너스를 추가할 필요는 없습니다. 자신의 보너스 함수들이 포함된 파일에만 _bonus를 추가하세요.

```
typedef struct      s_list
{
    void            *content;
    struct s_list   *next;
}                   t_list;
```

여기 t_list 구조체의 필드에 대한 설명이 있습니다.

- `content` : 요소에 포함된 데이터. Void 포인터는 어떠한 종류의 자료형이든 저장할 수 있습니다.
- `next` : 마지막 요소인 경우에는 NULL. or 다음 요소의 주소.

아래의 함수들은 당신의 리스트를 쉽게 사용할 수 있게 해줄 것입니다.

Function name	ft_lstnew
Prototype	t_list *ft_lstnew(void *content);
Turn in files	-
Parameters	#1. The content to create the new element with.
Return value	The new element.
External functs.	malloc
Description	Allocates (with malloc(3)) and returns a new element. The variable 'content' is initialized with the value of the parameter 'content'. The variable 'next' is initialized to NULL.

Function name	ft_lstadd_front
Prototype	void ft_lstadd_front(t_list **lst, t_list *new);
Turn in files	-
Parameters	#1. The address of a pointer to the first link of a list. #2. The address of a pointer to the element to be added to the list.
Return value	None
External functs.	None
Description	Adds the element 'new' at the beginning of the list.

Function name	ft_lstsize
Prototype	int ft_lstsize(t_list *lst);
Turn in files	-
Parameters	#1. The beginning of the list.
Return value	Length of the list.
External functs.	None
Description	Counts the number of elements in a list.

Function name	ft_lstlast
Prototype	t_list *ft_lstlast(t_list *lst);
Turn in files	-
Parameters	#1. The beginning of the list.
Return value	Last element of the list.
External functs.	None
Description	Returns the last element of the list.

Function name	ft_lstadd_back
Prototype	void ft_lstadd_back(t_list **lst, t_list *new);
Turn in files	-
Parameters	#1. The address of a pointer to the first link of a list. #2. The address of a pointer to the element to be added to the list.
Return value	None
External functs.	None
Description	Adds the element 'new' at the end of the list.

Function name	ft_lstdelone
Prototype	void ft_lstdelone(t_list *lst, void (*del)(void *));
Turn in files	-
Parameters	#1. The element to free. #2. The address of the function used to delete the content.
Return value	None
External functs.	free
Description	Takes as a parameter an element and frees the memory of the element's content using the function 'del' given as a parameter and free the element. The memory of 'next' must not be freed.

Function name	ft_lstclear
Prototype	void ft_lstclear(t_list **lst, void (*del)(void *));
Turn in files	-
Parameters	#1. The adress of a pointer to an element. #2. The adress of the function used to delete the content of the element.
Return value	None
External functs.	free
Description	Deletes and frees the given element and every successor of that element, using the function 'del' and free(3). Finally, the pointer to the list must be set to NULL.

Function name	`ft_lstiter`
Prototype	`void ft_lstiter(t_list *lst, void (*f)(void *));`
Turn in files	-
Parameters	#1. The adress of a pointer to an element. #2. The adress of the function used to iterate on the list.
Return value	None
External functs.	None
Description	Iterates the list 'lst' and applies the function 'f' to the content of each element.

Function name	`ft_lstmap`
Prototype	`t_list *ft_lstmap(t_list *lst, void *(*f)(void *), void (*del)(void *));`
Turn in files	-
Parameters	#1. The adress of a pointer to an element. #2. The adress of the function used to iterate on the list. #3. The adress of the function used to delete the content of an element if needed.
Return value	The new list. NULL if the allocation fails.
External functs.	malloc, free
Description	Iterates the list 'lst' and applies the function 'f' to the content of each element. Creates a new list resulting of the successive applications of the function 'f'. The 'del' function is used to delete the content of an element if needed.

당신이 적합하다고 생각한다면 당신의 libft에 어떠한 함수들을 자유롭게 추가할 수 있습니다.