Libft

# Part 1 - Libc functions

int ft\_isalpha(int c)

{

return (('a' <= c && c <= 'z') || ('A' <= c && c <= 'Z'));

}

int ft\_isdigit(int c)

{

return (c >= '0' && c <= '9');

}

int ft\_isalnum(int c)

{

return (ft\_isalpha(c) || ft\_isdigit(c));

}

int ft\_isascii(int c)

{

return (0 <= c && c <= 127);

}

int ft\_isprint(int c)

{

return (32 <= c && c <= 126);

}

size\_t ft\_strlen(const char \*s)

{

size\_t i;

i = 0;

while (s[i])

i++;

return (i);

}

void \*ft\_memset(void \*dst, int c, size\_t length)

{

size\_t i;

i = 0;

while (i < length)

{

((unsigned char \*)dst)[i] = (unsigned char)c;

i++;

}

return (dst);

}

void ft\_bzero(void \*dst, size\_t n)

{

ft\_memset(dst, 0, n);

}

void \*ft\_memcpy(void \*dst, const void \*src, size\_t length)

{

size\_t i;

if (!dst && !src)

return (dst);

i = 0;

while (i < length)

{

((unsigned char \*)dst)[i] = ((unsigned char \*)src)[i];

i++;

}

return (dst);

}

void \*ft\_memmove(void \*dst, const void \*src, size\_t length)

{

unsigned char \*usrc;

unsigned char \*udst;

if (dst == src || length == 0)

return (dst);

if (dst < src)

{

udst = (unsigned char \*)dst;

usrc = (unsigned char \*)src;

while (length--)

\*udst++ = \*usrc++;

}

else

{

udst = (unsigned char \*)dst + (length - 1);

usrc = (unsigned char \*)src + (length - 1);

while (length--)

\*udst-- = \*usrc--;

}

return (dst);

}

size\_t ft\_strlcpy(char \*dst, const char \*src, size\_t dstsize)

{

size\_t srcsize;

size\_t i;

if (dst == NULL && src == NULL)

return (0);

srcsize = ft\_strlen(src);

i = 0;

while (i < srcsize && i + 1 < dstsize)

{

dst[i] = src[i];

i++;

}

if (dstsize > 0)

dst[i] = 0;

return (srcsize);

}

size\_t ft\_strlcat(char \*dst, const char \*src, size\_t dstsize)

{

size\_t dst\_len;

size\_t src\_len;

size\_t i;

size\_t keep\_d;

keep\_d = 0;

while (keep\_d < dstsize && dst[keep\_d])

keep\_d++;

src\_len = ft\_strlen(src);

dst\_len = ft\_strlen(dst);

i = 0;

if (dstsize == 0 || dstsize < dst\_len)

return (src\_len + dstsize);

while (i < src\_len && dst\_len + i + 1 < dstsize)

{

dst[dst\_len + i] = src[i];

i++;

}

dst[dst\_len + i] = 0;

return (keep\_d + src\_len);

}

int ft\_toupper(int c)

{

if ('a' <= c && c <= 'z')

return (c - 32);

else

return (c);

}

int ft\_tolower(int c)

{

if ('A' <= c && c <= 'Z')

return (c + 32);

else

return (c);

}

char \*ft\_strchr(const char \*s, int c)

{

size\_t i;

i = 0;

if (c == 0)

return ((char \*)s + ft\_strlen(s));

while (s[i])

{

if (s[i] == (char)c)

return ((char \*)s + i);

i++;

}

return (0);

}

char \*ft\_strrchr(const char \*s, int c)

{

size\_t i;

i = ft\_strlen(s);

while (i > 0)

{

if (s[i] == (char)c)

return ((char \*)s + i);

i--;

}

if (s[i] == (char)c)

return ((char \*)s);

return (0);

}

int ft\_strncmp(const char \*s1, const char \*s2, size\_t n)

{

size\_t i;

i = 0;

while (i < n && (s1[i] || s2[i]))

{

if (s1[i] != s2[i])

return ((unsigned char)s1[i] - (unsigned char)s2[i]);

i++;

}

return (0);

}

void \*ft\_memchr(const void \*ptr, int value, size\_t num)

{

while (num--)

{

if (\*(unsigned char \*)ptr == (unsigned char)value)

return ((void \*)ptr);

ptr++;

}

return (0);

}

int ft\_memcmp(const void \*ptr1, const void \*ptr2, size\_t num)

{

size\_t i;

i = 0;

while (i < num)

{

if (((unsigned char \*)ptr1)[i] != ((unsigned char \*)ptr2)[i])

return (((unsigned char \*)ptr1)[i]

- ((unsigned char \*)ptr2)[i]);

i++;

}

return (0);

}

char \*ft\_strnstr(const char \*src, const char \*find, size\_t length)

{

size\_t i;

size\_t j;

if (find[0] == 0)

return ((char \*)src);

i = 0;

while (src[i] && i < length)

{

j = 0;

while (src[i + j] == find[j] && i + j < length)

{

j++;

if (find[j] == 0)

return ((char \*)src + i);

}

i++;

}

return (0);

}

int ft\_atoi(const char \*str)

{

long sign;

long nbr;

size\_t i;

i = 0;

while (str[i] == '\t' || str[i] == '\n' || str[i] == '\v'

|| str[i] == '\f' || str[i] == '\r' || str[i] == ' ')

i++;

sign = 1;

if (str[i] == '-' || str[i] == '+')

{

if (str[i] == '-')

sign = -sign;

i++;

}

nbr = 0;

while ('0' <= str[i] && str[i] <= '9')

{

nbr = nbr \* 10 + (str[i] - '0');

i++;

}

return (sign \* nbr);

}

void \*ft\_calloc(size\_t n, size\_t size)

{

void \*mem;

mem = (void \*)malloc(n \* size);

if (mem == 0)

return (0);

ft\_memset(mem, 0, n \* size);

return (mem);

}

char \*ft\_strdup(const char \*src)

{

char \*dup;

size\_t len;

len = 0;

while (src[len])

len++;

dup = (char \*)malloc(len + 1);

if (!(dup))

return (NULL);

ft\_memcpy(dup, src, len);

dup[len] = 0;

return (dup);

}

# Part 2 - Additional functions

## char \*ft\_substr(char const \*s, unsigned int start, size\_t len)

## {

## size\_t s\_size;

## char \*sub\_str;

## if (s == NULL)

## return (NULL);

## s\_size = (unsigned int)ft\_strlen(s);

## if (s\_size < start)

## return (ft\_strdup(""));

## if (ft\_strlen(s + start) < len)

## len = ft\_strlen(s + start);

## sub\_str = (char \*)malloc(sizeof(char) \* (len + 1));

## if (!sub\_str)

## return (NULL);

## ft\_strlcpy(sub\_str, s + start, len + 1);

## return (sub\_str);

## }

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

## {

## char \*new\_str;

## int s1\_len;

## int s2\_len;

## if (!(s1) && !(s2))

## return (NULL);

## else if (!(s1) || !(s2))

## {

## if (!(s1))

## return (ft\_strdup(s2));

## else

## return (ft\_strdup(s1));

## }

## s1\_len = ft\_strlen(s1);

## s2\_len = ft\_strlen(s2);

## new\_str = (char \*)malloc(sizeof(char) \* (s1\_len + s2\_len + 1));

## if (!(new\_str))

## return (NULL);

## ft\_strlcpy(new\_str, s1, s1\_len + 1);

## ft\_strlcat(new\_str + (s1\_len), s2, s2\_len + 1);

## return (new\_str);

## }

## char \*ft\_strtrim(char const \*s1, char const \*set)

## {

## size\_t i;

## if (!s1 || !set)

## return (NULL);

## while (\*s1 && ft\_strchr(set, \*s1))

## s1++;

## i = ft\_strlen(s1);

## while (i && ft\_strchr(set, s1[i]))

## --i;

## return (ft\_substr(s1, 0, i + 1));

## }

## static char \*\*ft\_malloc\_error(char \*\*res)

## {

## unsigned int i;

## i = 0;

## while (res[i])

## {

## free(res[i]);

## i++;

## }

## free(res);

## return (NULL);

## }

## static unsigned int word\_count(char const \*s, char c)

## {

## unsigned int i;

## unsigned int cnt;

## if (!s[0])

## return (0);

## i = 0;

## cnt = 0;

## while (s[i] && s[i] == c)

## i++;

## while (s[i])

## {

## if (s[i] == c)

## {

## cnt++;

## while (s[i] && s[i] == c)

## i++;

## continue ;

## }

## i++;

## }

## if (s[i - 1] != c)

## cnt++;

## return (cnt);

## }

## static void put\_res(char \*\*src, unsigned int \*len\_of\_w, char c)

## {

## unsigned int i;

## \*src += \*len\_of\_w;

## \*len\_of\_w = 0;

## i = 0;

## while (\*\*src && \*\*src == c)

## (\*src)++;

## while ((\*src)[i])

## {

## if ((\*src)[i] == c)

## return ;

## (\*len\_of\_w)++;

## i++;

## }

## }

## char \*\*ft\_split(char const \*s, char c)

## {

## char \*\*res;

## char \*src;

## unsigned int len\_of\_w;

## unsigned int i;

## if (!s)

## return (NULL);

## res = (char \*\*)malloc(sizeof(char \*) \* (word\_count(s, c) + 1));

## if (!res)

## return (NULL);

## i = 0;

## src = (char \*)s;

## len\_of\_w = 0;

## while (i < word\_count(s, c))

## {

## put\_res(&src, &len\_of\_w, c);

## res[i] = (char \*)malloc(sizeof(char) \* (len\_of\_w + 1));

## if (!res)

## return (ft\_malloc\_error(res));

## ft\_strlcpy(res[i], src, len\_of\_w + 1);

## i++;

## }

## res[i] = NULL;

## return (res);

## }

## int get\_len(int n)

## {

## int len;

## len = 0;

## if (n <= 0)

## len++;

## while (n != 0)

## {

## n = n / 10;

## len++;

## }

## return (len);

## }

## void ft\_fill\_res(int len, int flag, int n, char \*res)

## {

## while (len > flag)

## {

## res[len - 1] = n % 10 + '0';

## n = n / 10;

## len--;

## }

## }

## char \*ft\_itoa(int n)

## {

## char \*res;

## int flag;

## int len;

## flag = 0;

## len = get\_len(n);

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

## if (!res)

## return (0);

## if (n == -2147483648)

## {

## res[0] = '-';

## res[1] = '2';

## n = 147483648;

## flag = 2;

## }

## if (n < 0)

## {

## res[0] = '-';

## flag = 1;

## n = -n;

## }

## ft\_fill\_res(len, flag, n, res);

## res[len] = '\0';

## return (res);

## }

## char \*ft\_strmapi(char const \*s, char (\*f)(unsigned int, char))

## {

## unsigned int i;

## char \*res;

## if (!s || !f)

## return (NULL);

## res = (char \*)malloc(sizeof(char) \* ft\_strlen(s) + 1);

## if (!res)

## return (NULL);

## i = 0;

## while (s[i])

## {

## res[i] = f(i, s[i]);

## i++;

## }

## res[i] = '\0';

## return (res);

## }

## void ft\_striteri(char \*s, void (\*f)(unsigned int, char \*))

## {

## unsigned int i;

## if (!s || !f)

## return ;

## i = 0;

## while (s[i])

## {

## f(i, &s[i]);

## i++;

## }

## }

## void ft\_putchar\_fd(char c, int fd)

## {

## if (fd < 0)

## return ;

## write(fd, &c, 1);

## }

## void ft\_putstr\_fd(char \*s, int fd)

## {

## if (!s || fd < 0)

## return ;

## write(fd, s, ft\_strlen(s));

## }

## void ft\_putendl\_fd(char \*s, int fd)

## {

## if (!s || fd < 0)

## return ;

## write(fd, s, ft\_strlen(s));

## write(fd, "\n", 1);

## }

## void ft\_putnbr\_fd(int n, int fd)

## {

## if (fd < 0)

## return ;

## if (n == -2147483648)

## {

## ft\_putnbr\_fd(n / 10, fd);

## ft\_putchar\_fd(8 + '0', fd);

## }

## else if (n < 0)

## {

## ft\_putchar\_fd('-', fd);

## ft\_putnbr\_fd(-n, fd);

## }

## else

## {

## if (n > 9)

## ft\_putnbr\_fd(n / 10, fd);

## ft\_putchar\_fd(n % 10 + '0', fd);

## }

## }

## Bonus part

t\_list \*ft\_lstnew(void \*content)

{

t\_list \*new;

new = (t\_list \*)malloc(sizeof(t\_list));

if (new == NULL)

return (NULL);

new->content = content;

new->next = NULL;

return (new);

}

void ft\_lstadd\_front(t\_list \*\*lst, t\_list \*new)

{

if (lst == NULL || new == NULL)

return ;

new->next = \*lst;

\*lst = new;

}

int ft\_lstsize(t\_list \*lst)

{

int size;

size = 0;

while (lst != NULL)

{

lst = lst->next;

size++;

}

return (size);

}

t\_list \*ft\_lstlast(t\_list \*lst)

{

if (lst == NULL)

return (NULL);

while (lst->next != NULL)

lst = lst->next;

return (lst);

}

void ft\_lstadd\_back(t\_list \*\*lst, t\_list \*new)

{

t\_list \*last;

if (lst == NULL || new == NULL)

return ;

if (\*lst == NULL)

{

\*lst = new;

return ;

}

last = ft\_lstlast(\*lst);

if (last)

last->next = new;

}

void ft\_lstdelone(t\_list \*lst, void (\*del)(void \*))

{

if (lst == NULL || del == NULL)

return ;

del(lst->content);

free(lst);

}

void ft\_lstclear(t\_list \*\*lst, void (\*del)(void \*))

{

t\_list \*curr;

t\_list \*next;

curr = \*lst;

while (curr)

{

next = curr->next;

ft\_lstdelone(curr, del);

curr = next;

}

\*lst = NULL;

}

void ft\_lstiter(t\_list \*lst, void (\*f)(void \*))

{

if (lst == NULL || f == NULL)

return ;

while (lst)

{

f(lst->content);

lst = lst->next;

}

}

t\_list \*ft\_lstmap(t\_list \*lst, void \*(\*f)(void \*), void (\*del)(void \*))

{

t\_list \*result;

t\_list \*temp;

if (lst == NULL || f == NULL)

return (NULL);

result = NULL;

while (lst)

{

temp = ft\_lstnew((\*f)(lst->content));

if (!temp)

{

ft\_lstclear(&result, del);

return (NULL);

}

ft\_lstadd\_back(&result, temp);

temp = temp->next;

lst = lst->next;

}

return (result);

}