UNIVERSITY OF MALTA
FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

CPS1012: Operating Systems and Systems Programming I
Tutorial Sheet III - Memory Management

Author(s): Keith Bugeja
Tutor(s): Mark Magro, Josef Magri

Instructions:

1. Make sure you go through your course notes / slides before attempting the exercises.
2. The Unix **man** command is your best friend, **Google search** your second best.
3. Always test function return values for errors; report errors to the standard error stream.

Section A — *This tutorial is about memory management and string manipulation from first principles.*

When working out these examples, do not use the C library string manipulation functions, as this would defeat the purpose of the exercise.

You are to provide a function, `void string_free(char **p_str)` which frees allocated string buffers and sets `p_str` to `NULL`. An example of its usage follows:

```
char *my_string = string_copy("This is a string", NULL, 0);
...
string_free(&my_string);
```

1. Basic dynamic memory allocation

(a) Write a wrapper function for `malloc` that outputs an error if the allocation fails.
`void *xmalloc(size_t p_size, bool p_fatal);`

`p_size` specifies allocation request size in bytes;

`p_fatal` terminates program on error if set to `true`;

`returns` a `void*` to the allocated block; on error, if `p_fatal == false`, the function returns `NULL`.

(b) Write a function similar to (a) above that zeroes allocated memory on request:
`char *string_alloc(size_t p_size, bool p_clear, bool p_fatal);`

`p_size` specifies allocation request size in bytes;

`p_clear` returns a cleared (zeroed) memory block if set to `true`;

`p_fatal` terminates program on error if set to `true`;

`returns` a `char*` to the allocated block; on error, if `p_fatal == false`, the function returns `NULL`.

(c) Write a function that releases allocated memory and sets the respective pointer to `NULL`:
`void string_free(char **p_str);` (see section information above)

`p_str` address of pointer to memory block

2. String manipulation

(a) Implement a function that returns the length of a `NULL`-terminated string; assume the function has the following signature: `size_t string_length(const char *p_str);` where

`p_str` is a pointer to a `NULL`-terminated string;

`returns` the length of the string **not** including the `NULL` terminator.

(b) Implement a function that copies a `NULL`-terminated string to a specified destination buffer:
`char *string_copy(const char *p_src, char *p_dst, size_t p_size);` where

p_src is a pointer to the NULL-terminated source string;

p_dst is a pointer to the destination buffer;

- if p_dst == NULL or string_length(p_src) > p_size, allocate (or reallocate) enough memory for the destination buffer to hold the entire source string

p_size specifies the current size of the destination buffer.

returns p_dst on a successful copy without allocation (or reallocation); otherwise, return a pointer to the newly allocated memory block. On error, return NULL.

(c) Extend (b) to allow copying substrings of the source:

```
char *string_sub(const char *p_src, int p_s, int p_e, char *p_dst, size_t
↪  p_size);
```

p_src is a pointer to the NULL-terminated source string;

p_s, p_e denote the starting and ending character indices of the substring;

p_dst is a pointer to the destination buffer;

- if p_dst == NULL or 1 + p_e - p_s > p_size, allocate (or reallocate) enough memory for the destination buffer to hold the entire source string

p_size specifies the current size of the destination buffer.

returns p_dst on a successful copy without allocation (or reallocation); otherwise, return a pointer to the newly allocated memory block. On error, return NULL.

(d) Write a simple search function that finds the first occurrence of the character p_c within a NULL-terminated string: `char *string_find(const char *p_src, char p_c);`

p_src is a pointer to the NULL-terminated source string;

p_c is the character to search for;

returns pointer to first occurrence of p_c in p_str. If no occurrences are found, the function returns NULL.

3. Scanning and parsing

(a) Write a function that splits a NULL-terminated string into a number of substrings. Substrings are delimited by the specified character p_delim:

```
char **string_split(const char *p_str, char p_delim);
```

p_str is a pointer to a NULL-terminated string;

p_delim is a character delimiter used to split the source string into multiple strings;

returns an array of strings comprised of the substrings of p_str delimited by p_delim. Note that the function should allocate memory for each substring in the list, which has NULL as its last entry.

(b) Extend (a) to provide two additional delimiters, `p_left` and `p_right` wherein `p_delim` occurrences are ignored.

```
char **string_split_ex(const char *p_str, char p_delim, char p_left, char
↪  p_right);
```

p_str is a pointer to a `NULL`-terminated string;

p_delim is a character delimiter used to split the source string into multiple strings;

p_left, p_right are two delimiter characters denoting portions of the string wherein `p_delim` is ignored:

```
// calling:
string_split_ex("This is [a test string for split_ex] ! Spaces are
↪  [ignored here as well.]", ' ', '[', ']');
// returns the elements:
// This,is,[a test string for split_ex],!,Spaces,are,[ignored here as
↪  well.]
```

returns an array of strings comprised of the substrings of `p_str` delimited by `p_delim`. Note that the function should allocate memory for each substring in the list, which has `NULL` as its last entry.

(c) Implement a string evaluation function that that replaces substrings satisfying the regular expression `$[0-9a-zA-Z_]*` with the respective environment variable, if the latter exists:

```
char *string_evaluate(const char *p_str, char *p_dst, size_t p_size);
```

p_src is a pointer to the `NULL`-terminated source string;

p_dst is a pointer to the destination buffer, where the string with replacements is stored;

- if `p_dst == NULL` or the replacement string is larger than `p_size`, allocate (or reallocate) enough memory for the destination buffer to hold the entire result

p_size specifies the current size of the destination buffer.

returns `p_dst` on a successful copy without allocation (or reallocation); otherwise, return a pointer to the newly allocated memory block. On error, return `NULL`.

```
// assume $HOME=/home/student and $USER=student
char s1 = NULL, s2 = "$USER lives at $HOME.";
s1 = string_evaluate(s2, NULL, 0);
...
printf("[%s]\n", s1);
// outputs: student lives at /home/student.
...
string_free(s1);
```