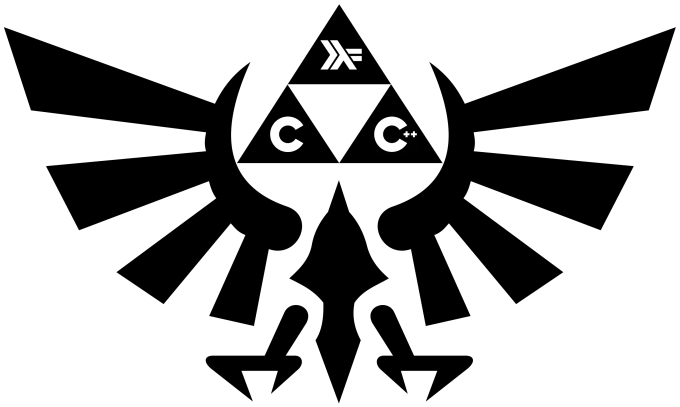
B3 - Paradigms Seminar

B-PDG-300

Day 05

String module



2.2

Day 05

binary name: libstring.a
language: C
compilation: via Makefile, including re, clean and fclean rules

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

All your exercises will be compiled with the `-std=gnu17 -Wall -Wextra` **flags**, unless specified otherwise.

All output goes to the standard output.

None of your files must contain a `main` function, unless specified otherwise.
We will use our own `main` functions to compile and test your code.

Your Makefile must build your source files and create a static library called `libstring.a`.
You must also submit a file called `string.h` containing the definition of your module (structure and init/destroy functions prototypes).

Read the examples CAREFULLY. They might require things that weren't mentioned in the subject…

Unit Tests

It is highly recommended to test your functions as you implement them. It is common practice to create and use what are called **unit tests**.

From now on, we expect you to write unit tests for your functions (when possible). To do so, please follow the instructions in the **"How to write Unit Tests"** document on the intranet, available here.

EXERCISE 0 - MY STRING

Create a `string_t` module.
The structure must hold a `char *str` member, and your module should contain an initialization function and a destruction function with the following prototypes:

```
void string_init(string_t *this, const char *s);
void string_destroy(string_t *this);
```

`string_init` assigns the value of `s` to the `str` member of the structure `string_t`.
Here is a sample `main` function and its expected output:

```
#include <stdio.h>
#include "string.h"

int main(void)
{
    string_t s;

    string_init(&s, "Hello World");
    printf("%s\n", s.str);
    string_destroy(&s);
    return (0);
}
```

Terminal

```
~/B-PDG-300> make
gcc test_main.c -L. -lstring
~/B-PDG-300> ./a.out
Hello World
```

EXERCISE 1 - ASSIGN

Add the following member functions to your module:

```
void assign_s(string_t *this, const string_t *str);
```

Sets the content of the current instance to that of `str`.

```
void assign_c(string_t *this, const char *s);
```

Sets the content of the current instance to `s`.

Remember to assign your function pointers.

Be careful with memory leaks.

Here is a sample `main` function and its expected output:

```
#include <stdio.h>
#include "string.h"

int main(void)
{
    string_t s;

    string_init(&s, "");
    s.assign_c(&s, "Hello World");
    printf("%s\n", s.str);
    string_destroy(&s);
    return (0);
}
```

```
Terminal
~/B-PDG-300> make
gcc test_main.c -L. -lstring
~/B-PDG-300> ./a.out
Hello World
```

Exercise 2 - Append

Add the following member functions to your module:

```
void append_s(string_t *this, const string_t *ap);
```

Appends the content of `ap` to that of the current instance.

```
void append_c(string_t *this, const char *ap);
```

Appends `ap` to the content of the current instance.

Exercise 3 - At

Add the following member function to your module:

```
char at(const string_t *this, size_t pos);
```

Returns the `char` at the `pos` position of the current instance, or -1 if the position is invalid.

`'\0'` is not really part of the string.

Exercise 4 - Clear

Add the following member function to your module:

```
void clear(string_t *this);
```

Empties the content of the current instance.

Clearing the content means that the string is now empty and of size 0.

Exercise 5 - Length

Add the following member function to your module:

```
int length(const string_t *this);
```

Returns the size of the string, or -1 if the `str` pointer of the string is `NULL`.

Exercise 6 - Compare

Add the following member functions to your module:

```
int compare_s(const string_t *this, const string_t *str);
```

Compares the content of the current instance to that of `str`. Results are the same as the `strcmp` function.

```
int compare_c(const string_t *this, const char *str);
```

Compares the content of the current instance to `str`. Results are the same as the `strcmp` function.

Exercise 7 - Copy

Add the following member function to your module:

```
size_t copy(const string_t *this, char *s, size_t n, size_t pos);
```

Copies up to `n` characters from the current instance's content, starting from the `pos` position, into `s`. It returns the number of characters copied.

If there's no null byte in the n'th characters copied, none is added to `s`.

Exercise 8 - c_str

Add the following member function to your module:

```
const char *c_str(const string_t *this);
```

Returns the buffer contained in the current instance.

Exercise 9 - Empty

Add the following member function to your module:

```
int empty(const string_t *this);
```

Returns 1 if the string is empty, 0 otherwise.

Exercise 10 - Find

Add the following member functions to your module:

```
int find_s(const string_t *this, const string_t *str, size_t pos);
```

Searches for the first occurence of `str`'s content in the current instance, starting from the `pos` position.

```
int find_c(const string_t *this, const char *str, size_t pos);
```

Searches for the first occurence of `str` in the current instance, starting from the `pos` position.

These functions return the position (relative to the beginning of the string) where the occurence was found, or -1 if `str` wasn't found, if `str` is too long or if the position is invalid.

Exercise 11 - Insert

Add the following member functions to your module:

```
void insert_c(string_t *this, size_t pos, const char *str);
```

Copies `str` into the current instance, at the `pos` position.

```
void insert_s(string_t *this, size_t pos, const string_t *str);
```

Copies the content of `str` into the current instance, at the `pos` position.

These functions enlarge the current instance, meaning the content is inserted at `pos` but doesn't replace the current content of our string. If `pos` is greater than the size of the current instance, `str` should be appended to its content.

Be careful with null-terminating bytes…

Exercise 12 - To_int

Add the following member function to your module:

```
int to_int(const string_t *this);
```

Returns the content of the current instance converted into an int. It behaves like the `atoi(3)` function.

Exercise 13 - Split

Add the following member functions to your module:

```
string_t **split_s(const string_t *this, char separator);
```

Returns an array of pointer to strings filled with the content of the current instance split using the `separator` delimiter.

```
char **split_c(const string_t *this, char separator);
```

Returns an array of C-style strings filled with the content of the current instance split using the `separator` delimiter. Each string of the array should be individually allocated.

Two consecutive delimiters produce an empty string.

Each element of the array must be individually allocated. The last element of the returned array must be `NULL`.

Exercise 14 - Print

Add the following member function to your module:

```
void print(const string_t *this);
```

Displays the content of the current instance to the standard output.

Be careful, we never said anything about carriage returns!

Exercise 15 - Join

Add the following member functions to your module:

```
void join_c(string_t *this, char delim, const char * const * array);
```

Assigns a string of characters created by joining all the C-style strings in `array`, separated by the `delim` delimiter, to the current instance. `array` will always be null-terminated.

```
void join_s(string_t *this, char delim, const string_t * const * array);
```

Assigns a string of characters created by joining all the strings in `array`, separated by the `delim` delimiter, to the current instance. `array` will always be null-terminated.

Exercise 16 - Substr

Add the following member function to your module:

```
string_t *substr(const string_t *this, int offset, int length);
```

Extracts a substring of `length` characters from the current instance, starting from the `offset` position. It returns the substring as a new `string_t` instance.

If `offset` is negative, it must be interpreted as the number of characters to skip starting from the end.
If `length` is negative, it represents the number of characters to be copied from the left of the `offset`.
If these specifications are in part out of bounds, the generated substring must be truncated to only contain parts of the current instance.