# SW and HW functional verification - Home assignment

## Introduction

The scope of the home assignment is for candidates to show programming skills in C and Python. The code should exhibit production quality. In case of ambiguity in the assignment, the candidate is free to take any assumption.

In return the candidate shall send a compressed file with all relevant files.

## Circular buffer in C

Implement queue based on circular buffer. A circular buffer is based on an array of fixed size which is virtually connected end to end, i.e. when the end of the array is reached, writing is done at the start of the array. In our case as the buffer is used for a queue, when the maximum capacity is reached, an error code should be returned when trying to add more elements to the queue. Attempt to dequeue an element from an empty queue should also result in an error.

Write unit test cases for the queue. The following API should be implemented:

**Queue API**

```
#define QUEUE_IS_EMPTY      1
#define QUEUE_IS_FULL       2

typedef struct queue queue_t;

void init(queue_t* q);
int  enqueue(queue_t* q, int val);
int  dequeue(queue_t* q, int* val);
bool is_empty(queue_t* q);
```

## Log parser in Python

### Background

A continuous integration test system for an embedded software stack is running test cases on several build servers where each of them has special hardware connected. There is parallel execution of the test cases among the build servers. Test results from each build server are copied to shared file repository.

The output of a test execution will be log files and several .xml-files containing detailed test results for each test case. Some of the test cases are grouped into test suites, hence a single .xml-file can have test results for several test cases.

**XML log example**

```
<test_results test_suite="GAP_CONN_001">
  <environment> ... </environment>
  <debug> ... </debug>
  <tc_result id="conn_bv_001" result="PASS">
    <debug> ... </debug>
  </tc_result>
  <tc_result id="conn_bv_002" result="FAIL" />
  <tc_result id="conn_bv_004" result="SKIP">
    <reason> ... </reason>
  </tc_result>
  ...
</test_results>
```

**Folder contents example**

```
.../test/gap/conn/gap_conn_001.xml
.../test/gap/sec/seci/gap_sec_001.xml
.../test/gap/sec/seci/gap_sec_002.xml
.../test/gap/rssi/gap_rssi_001.xml
.../test/gatts/gaw/gatts_gaw_001.xml
```

**Task**

Develop an XML Log parser in Python that will read all XML-files from a specified folder and generate a test report with statistics for the complete test execution.

Requirements:

- The tool shall have a command line interface
- Input parameter shall be the root folder containing XML-files in sub-folders
- The parser itself shall be a class derived from AbstractLogsParser (see below)
- The tool shall provide metrics for summary and detailed report
- Add test cases to verify the implementation

**AbstractLogsParser**

```
class AbstractLogsParser(object):
	# Different test statuses
	TEST_RES_PASS = 0
	TEST_RES_FAIL = 1
	TEST_RES_SKIP = 2

	def __init__(self, logs_extension):
		"""
		Base class constructor.
		@param	logs_extension  Extension of log files to parse.
		"""
		self._logs_ext = '.'+logs_extension

	def get_result_by_type(self, result_type):
		"""
		Returns number of passed, failed or skipped tests.
		@param	result_type		Type of results to return.
		"""
		return -1

	def generate_detailed_report(self):
		"""
		Generates detailed report on each test suite.
		"""
		raise Exception("generate_detailed_report is not implemented")

	def process_logs(self, folder):
		"""
		Parses all log files with target extension in the specified folder.
		@param	folder		Folder to look up for log files.
		"""
		raise Exception("process_logs is not implemented")
```

The following example XML log files shall be used to verify the program (note the folders structure with 'test' being the root folder):

**Log files**

```
./test/gap/conn/gap_conn_001.xml

<test_results test_suite="GAP_CONN_001">
  <environment>LOCAL</environment>
  <debug>Test suite start</debug>
  <tc_result id="conn_bv_001" result="PASS">
    <debug>Run 2 sequences</debug>
  </tc_result>
  <tc_result id="conn_bv_002" result="FAIL" />
  <tc_result id="conn_bv_002" result="PASS" />
  <tc_result id="conn_bv_004" result="SKIP">
    <reason>No pass criteria</reason>
  </tc_result>
</test_results>

./test/gap/sec/seci/gap_sec_001.xml

<test_results test_suite="GAP_SEC_001">
  <environment>SERVER002</environment>
  <tc_result id="sec_auth_001" result="PASS" />
  <tc_result id="sec_auth_002" result="SKIP" >
    <reason>Unstable</reason>
  </tc_result>
  <tc_result id="sec_auth_003" result="PASS" >
    <debug>Auth on 2nd attempt</debug>
  </tc_result>
  <tc_result id="sec_auth_004" result="PASS" />
</test_results>

./test/gap/rssi/gap_rssi_001.xml

<test_results test_suite="GAP_RSSI_001">
  <environment>LOCAL</environment>
  <tc_result id="gap_rssi_001" result="PASS" />
  <tc_result id="gap_rssi_002" result="FAIL" >
    <debug>RX 10 dBm</debug>
  </tc_result>
  <tc_result id="gap_rssi_003" result="PASS" />
  <tc_result id="gap_rssi_004" result="SKIP" >
    <reason>Not implemented</reason>
  </tc_result>
  <tc_result id="gap_rssi_005" result="SKIP" >
    <reason>Not supported</reason>
  </tc_result>
</test_results>
```

End of document