COMP20003 Assignment 1

1	understanding the task
2	considerations
3	strategies

Understanding: The Big Task = Programming + Report

Programming Tasks: build at least 3 modules, say:

- `tst.c` and `tst.h`: for working with ternary search trees
- `main1`.c: for using tst in Stage 1 (`autocomplete1`)
- `main2`.c: for using tst in Stage 2 (`autocomplete2`)

Why "at least 3"?

- There might be some extra functions you don't want to put in neither `tst.c` nor `main.c` …
- There might be some utilities (like `safe_malloc`, `safe_calloc`, `safe_fopen`, that you can even use for assignment 2. These tools can be combined into, say, `utils.c` and `utils.h`

Sample structure with 4 modules (= 6 files)

tst.h

```
#include …
typedef...
```

function prototypes

utils.h

```
#include …

void
*safe_malloc(int
zise);
```

other function prototypes

main1.c

```
#include “tst.h”
#include “utils.h”
...
```

… main(…) …

```
. . .
```

tst.c

```
#include
“tst.h”
```

function implementation

utils.c

```
#include
“utils.h”
```

function implementation

main1.c

```
#include “tst.h”
#include “utils.h”
...
```

… main(…) …

```
. . .
```

Makefile

Don’t forget `Makefile`, it’s a requirement!

In the Makefile, you should have 4 targets/goal:

- `all`: `autocomplete1 autocomplete2`
- `autocomplete1`: <dependencies & command…>
- `clean`:

“`all`” must be the first target. However, you should insert ‘`all`” into you `Makefile` only after having `autocomplete1` and `autocomplete` 2 working preperly!

Need example? See `Makefile` in `github.com:anhvir/c203`

Sample `Makefile`’s line for `autocomplete1`:

```
autocomple1: Makefile main1.c tst.c utils.c
   gcc –Wall -o autocomplete1 main1.c tst.c utils.c
```

Considerations:

- write and use `safe_malloc`, `safe_realloc`, `safe_fopen`. Alternatively, you should use `assert` after any `malloc`, `realloc`, and `fopen`.
- Couple `malloc` with `free`.
- Couple `fopen` with `fclose`.
- Add flag –g to `Makefile`, and run `gdb` to make sure that you have a clean report on memory usage! It's better to remove –g before running experiments.
- Use simple `scanf("%d;%[^\n]", &an_int, a_string)` for reading data, don't write a complicated code for that (unless you're a billionaire of time).
- `Google` is an excellent and handy friend!

Report (in PDF format)

Report:

- should have an intro clearly and briefly stating the purpose of the report;
- should have discussion and a short conclusion;
- should describe experiment methodology;
- should use graphs and/or tables for presenting data and discussion;
- should follow the guidelines in specs; and
- should be concise (not a long report)!

Strategies

Incremental development:

- write and test `main.c` without `tst.c`, make sure that you can handle the arguments correctly.
- build `tst.h` and `tst.c` with a single function insert, and test to make sure that insert work.
- add on 1 function at a time.
- build a test data file and use redirection, don't enter data each time you want to test your program.
- AND OF COURSE: make a careful timeline, remember the deadline.
- Test your program on `dimefox` or `nutmeg`!
- Submit frequently, after any working session, or after making a reasonable change!
- Reserve at least 1 day for experiment and report!