ASSIGNMENT SUBMISSION STANDARD
CST8152 – Compilers

Notation: [something] -> replace **[something]** with the required information
(-X%) -> percentage from your maximal mark for the assignment which will be deducted from your assignment mark if you do not comply with the standard.
Examples: Assignment:[number] -> Assignment: 1
(-5%) -> 5% from your maximal mark for the corresponding assignment

A. Submission (-5%)

Two types of assignment submission are required for this course: Assignment Box (AB) submission (printed paper) and Blackboard digital (BBD) electronic submission. The contents of the AB and BBD submissions will be outlined in the assignment submission section of each individual assignment.

The Assignment Box submission must be placed into an unsealed page-size envelope labeled with the following information:

[Student Name & ID number]
Course: CST 8152 – Compilers, **Lab Section:** [11, 12 or 13]
Assignment: [number – 1,2,3 …]
Professor: Sv. Ranev
Date: [submission date]

Note: Students who submit their assignments in transparent envelopes need not to label them. They must make sure that their cover page (see below) is visible.

B. Cover Page (-5%)

All printed assignments must have a cover page with the following information:

[Student Name & ID number**]**
Course: CST 8152 – Compilers, **Lab Section:** [11, 12 or 13]
Assignment: [number]
Professor: Sv.Ranev
Due Date: [due date]
Date: [submission date]
Contents: [you must list here all the files/printouts submitted; they must be listed in the same order as the order they are arranged in the envelope]

C. File (Program) Headers (-5% each)

Each of your **.h** or **.c** files must have a file header written as ANSI C89 comments and containing the following information:

File name: [my_masterpiece.h or my_masterpiece.c and so on]
Compiler: [MS Visual Studio 20XX, Borland 5.X, C++Builder, gcc, or other]
Author: [Student name, ID#]
Course: CST 8152 – Compilers, **Lab Section:** [11, 12 or 13]
Assignment: [number]
Date: [the date of the final version of the file]
Professor: Sv. Ranev
Purpose: [brief description of the contents]
Function list: [list here all the functions declared/defined in this file. Do not include the function parameter lists and return type, i.e. *func()*, not *int func(int a);* the list must follow the order of the function declaration/definition in the file]

D. Function headers (up to -10%)

Each of your function definitions must have a function header written as ANSI C89 comments and containing the following information:

Purpose: [briefly explain the purpose of the function]
Author: [name (your name if you wrote the function)]
History/Versions: [version numbers and dates]
Called functions: [list of function(s) that are called by this function]
Parameters: [for each formal parameter: type, specific range or values if applicable]
Return value: [type, specific values if applicable]
Algorithm: [outline the main steps (sections) only; do not include implementation details; for small and clear functions leave this empty]

E. Comments (up to -10%)

- All variable definitions/declarations must be commented. The comment must explain the use of the variable.
- All function segments must be commented (a segment is a sequence of related statement i.e. loops, switches, if-else ladders and sequence of linear statements performing some distinctive task.)
- Comment each important line of code. Use your judgment to decide whether the line is important. Important lines: testing some special conditions; dynamic allocation; complex calculations; conversions and so on. Do not comment overly your programs.

F. Test plan (up to -10%)

Include a brief description of your testing strategy and the expected results.

ASSIGNMENT MARKING GUIDE

CST8152 – Compilers

Working Program Definition

An assignment ANSI C program is a **working program** if and only if

- the program files (including the provided main program and some other .h or .c files) compile and link with no errors using the standard project for the **test-bed compiler Microsoft Visual Studio 2015**;
- the program runs as a console application and does not crash at run-time when processing the test files;
- the program complies with the **functional specifications** given in the assignment. The functional specifications include names, types, prototypes, and functionality;
- the program produces the expected output when processing the input data/files specified in the assignment. If output files are provided for the assignment, the program must produce **byte by byte identical** output files when it is run with the corresponding test files. Adjustments to the program, which violate the specifications, and which are implemented with the sole purpose of producing an identical output will render the program non-working.

The following set of marking rules will be applied when I calculate your assignment marks.

RYGP. "You are Great and Perfect" Rule

At the beginning you receive 100% of the maximal mark for the assignment.

And then some deductions might apply.

RBGO. "Big Giant 0" Rule

If you do not submit a **working program** (see the definition above), you automatically turn into a winner of the "Big Giant 0" award. I might still give you some credits but your final mark will not exceed 55% of the maximal mark for the assignment.

RWI. "What IF" Rule

I will test your program thoroughly with a set of additional test files trying to find all program "deficiencies." If your program passes all the tests, I will apply **RYGP**.
If I find some problems, depending on the problem I will apply some deductions but no more than 40% of the maximal mark.

RTIAOMB. “There is Always One More Bug” Rule

I will read your code carefully and if I find minor mistakes, I will apply some deductions but no more that 10% of the maximal mark. Minor mistakes are: insignificant specification violations, redundant code, unused code, unnecessary extra steps and alike.

RWAFR. “Warnings are for Reading” Rule

If your program compiles with warnings and those warnings are not justified and not explained in your comments, every warning will cost you 1%.

RIDTS. “I Detest the Standards” Rule

This rule will be applied if your assignment does not comply with the assignment submission standard outlined in the Assignment Submission Standard document. The deductions are listed next to the corresponding requirements.

RIAL. “I am Late” Rule

In order to pass the course and receive a full credit, all assignment must be submitted on time. Due to the specific nature of the course evaluation – deferred success (in other words F) if the complete program is not working at the end – late assignment submission will be allowed but there will be significant mark deductions. The deductions for lateness are: **20%** of the assignment allocated marks for a submission in the first week after the due date; **50%** in the second week; **100%** for beyond two weeks late submission. All assignments must be completed in order to pass the course, even if the assignments are late. The last assignment can not be late.

RITN. “Isn’t that nice” Rule

You can get up to 10% for clever and clear coding, and useful enhancements.

RABGO. “Another Big Giant O” Rule

The code of the assignment must be originally written by the student submitting the assignment. If it happens that you have “borrowed” the entire code or even parts of the code, you will get a Big Giant **0**. And something more, see the course outline. And I am serious. And make no mistake about it…

In conclusion, here is a regular expression describing your assignment final mark (FM):

FM -> **(RYGP** | **RYGP – RWI? – RTIAOMB? – RWAFR? – RIDTS? – RIAL?)+RITN?**
| RBGO + SC?
| RABGO

SC -> some credits but not more than 55%

Enjoy the course and do not forget that:

“Experience is the worst teacher; it gives the test before presenting the lesson.”
Vernon Law

But

“There is no substitution for experience”

And remember to remember:

“Not everything that counts can be counted and not everything that can be counted counts.” Albert Einstein

And never forget the “Debugging Paradox”

“Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.”
-Brian W. Kernigan

The solution is careful design and implementation, and then testing, testing, and testing again. Once you bypass the testing step, no amount of debugging will save you. Always have a battery of tests written to prove that a function is working as specified, and use the debugger to help you figure out why a straightforward fragment of code is failing.
Of course, when you reach a point where all else fails, there is always the all-purpose `printf()`.