# CSC4180 Assignment 1: Design a compiler for Micro

```
119010181 Lin Xinhao
```

For both flex and bison, I mainly learnt their functions from this blog: https://blog.csdn.net/weixin_44007632/article/details/108666375, which I think is quite good in details.

## How did I design the Scanner?

I used flex to help me generate scanner by filling scanner.l. My scanner is really short, I only recognized the 14 tokens as decribed in the assignment description, and I ignored comments which are lines started with '--.' and ended with \n.

## How did I design the Parser?

Still, I used bison to help generate parser code. For the tokens passed from scanner, I set INTLITERAL(int) and ID(char[32]) to be binded with structs. I tried to use c++'s string type at first, but I failed. It is because here the %union doesn't supported string type. Then I just followed the instruction of the assignment with several type defined and they are: <id_list_struct> id_list, <prim_struct> primary, <expr_list_struct> expr_list, <expr_struct> expression, <add_op_struct> add_op. I simply converted CFG into LR and assigned each of the type node a function.

## How the code is generated?

This part is quite hard, I did these things:

1. I defined a tmp register table with all 10 tx registers to check their occupancy.
2. I defined a stack pointer which will always decrease when new tmp/varible things are stored.
3. I defined a map called ID_lookup_table which stores all the variable's name-memory_address pair.

Then I defined a lot of functions:

4. for low level MIPS generation, I gave each of the MIPS code a generation function (e.g.,mips_load_word(int target_reg,int pos_reg,int shift);//reg_number = MEM[reg_number+shift]), and all the code generation part need to use them.
5. I set a lot of statues checking function for register allocation and de-allocation, like: use_a_tmp_reg(), free_a_tmp_reg(int reg_num), free_all_tmp_reg(). And some functions for finding ID and storing ID.

In the end, my generation mechanism is that: I stored all the variable in memory (no $sx register is used), and passed all the address/register number/immediate int value to nodes' parent root, this process is repeatly done until it reaches an execution expression (in expression add_op primary), where it will generate code and store them in memory (again, I choose to store in memory instead of some register because when a expression or expression list is too complex, the registers will be used up easily). This part is the most difficult among all the codes.

Lastly, assign/read/write will do some simple MIPS instruction to make the statement complete.