

# COMP 3031 Assignment 2

## Flex and Bison Programming

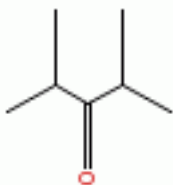
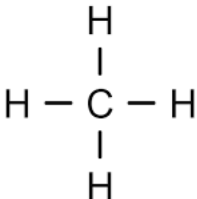
### Fall 2021

### Due 5pm, 5th Nov. 2021 Friday

## 1. Problem Description

In this assignment, you will implement a calculator for a simplified molecular-input line-entry system (SMILES) by filling in the blanks in the provided flex and bison code skeletons.

The SMILES is a string used to describe the structure of chemical species. For example:

Structure of chemical species	SMILES
	<chem>CC(C)C(=O)C(C)C</chem>
	<chem>[CH4]</chem>

The BNF grammar rules for SMILES are as follows:

```
<smiles> ::= <atom> | <atom><BC>
```

```
<BC> ::= <branch> | <chain> | <branch><BC> | <chain><BC>
```

```
<chain> ::= <atom> | <bond><atom> | <atom><chain> | <bond><atom><chain>
```

```
<branch> ::= (<chain>)
```

```
<atom> ::= <bracket_atom_substructure> | <aliphatic_organic> | <aromatic_organic>
```

```

<aliphatic_organic> ::= B | C | N | O | S | P | F | Cl | Br | I
<aromatic_organic> ::= b | c | n | o | s | p
<bracket_atom_substructure> ::=
[<symbol>][<aliphatic_organic>][<symbol><hattached>][<aliphatic_organic><hattached>]
[<hattached>]
<symbol> ::= H | He | Li | Be | Ne | Na | Mg | Al | Si | Ar | K | Ca
<hattached> ::= H | H <digit>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<bond> ::= - | = | # | $ | / | \

```

There are 6 operators, “|”, “+”, “UNIQUE”, “MAX”, “MIN”, and the braces “{ }”.

The operators are listed in the order of decreasing precedence.

Operators	Operations	Associativity
{ }	Braces	
	Count the number of a given atom in the chemical species. The input atom should satisfy the grammar rules for SMILES.	Left
UNIQUE	Count the total number of unique atoms in the chemical species.	Right
MAX, MIN	The number of the most/least atom in the chemical species.	Right
+	Number addition.	Left

The following BNF grammar rules define expressions with SMILES operations:

```

<expression> ::= <smiles>
| <smiles> <count> <smiles>
| UNIQUE <smiles>
| MAX <smiles>
| MIN <smiles>
| <expression> <addition> <expression>

```

```

| <left_brace_bracket><expression><right_brace_bracket>
<count> ::= |
<addition> ::= +
<left_brace_bracket> ::= {
<right_brace_bracket> ::= }

```

For example:

Input: CCC=O|C

Output: 3

Input: [CH4]||[H]

Output: 4

Input: UNIQUE N[CH](C)C(=O)O

Output: 4

Input: MAX [CH4]

Output: 4

Input: {CCC=O|C}+{CCC(CC)CO|C}

Output: 9

## 2. Your work

We provide a zip package containing the following files:

assignment2.pdf	Assignment 2 description
helpers.h, helpers.c	Helper functions
Makefile	Makefile that supports “make” and “make clean”

SMILEScal.lex	Flex code skeleton for you to fill in
SMILEScal.y	Bison code skeleton for you to fill in

You can decompose the zip file using the following command

```
unzip assignment2-2021fall.zip
```

**Your tasks are:**

1. Add missing flex definitions and rules in SMILEScal.lex
2. Add missing tokens and grammar rules in SMILEScal.y

We have marked the blanks that you can fill in with comment lines:

```
/***** Start: ... *****/
```

```
/***** End: ... *****/
```

Note: Do **not** modify the provided files *Makefile*, *helpers.h* and *helpers.c*.

### 3. Compilation and Testing

After finishing the code, you can compile your code on a CS Lab 2 machine with the command

```
make
```

After successful compilation, you can run the binary, and type in the commands to start the SMILES calculator:

```
./SMILEScal
```

Your result SMILES calculator will work as follows:

- (1) If the input expression is a single SMILES, your calculator will print the number of atoms in the SMILES. For example,

```
CCC=0
```

```
4
```

- (2) If the input expression contains operators, your calculator will compute the result of the expression. For example,

```
{UNIQUE CCC=0}+{MAX CCC(CC)CO}
```

Your program does **not** need to handle input errors.

## 4. Helper functions

We list the helper functions in helpers.h and helpers.c in the following table:

Helper function	Description
void *generateSmiles(void* smilesPtr);	Generate a struct and return its pointer from the given smiles string, the struct needs to be passed to other helper functions.
void *count_atom(void* resultPtr1, void *resultPtr2);	Count the number of the given atom resultPtr2 appearing in smilesStruct resultPtr1.
void* max_atom(void *resultPtr);	Count the number of atoms that occurred the most in the smilesStruct resultPtr
void* min_atom(void* resultPtr);	Count the number of atoms that occurred the least in the smilesStruct resultPtr
void* count_all_atoms(void* resultPtr);	Count the total number of atoms in smilesStruct resultPtr
void* unique_atom(void* resultPtr);	Count the number of unique atoms in smilesStruct resultPtr
void *addNum(void* resultPtr1, void *resultPtr2);	Add two int numbers resultPtr1, resultPtr2 and return the result
void printElem(void *resultPtr);	Print the return value resultPtr from all other functions

## 5. Submission:

- Zip your two source files, SMILEScal.lex and SMILEScal.y, into a single package using exactly the following command (case-sensitive):

```
zip SMILEScal.zip SMILEScal.lex SMILEScal.y
```

- Submit your zipped file SMILEScal.zip to Canvas.
- **No late submission will be accepted.**
- Your submission will be compiled and run on a CS Lab 2 machine. If it cannot be compiled or run properly, you may get 0 marks for this assignment.
- We will use tools to detect code similarity. On confirmed cases of high code similarity, we will follow [university guidelines on academic integrity](#) to take necessary actions.