Final Case Study Documentation

Group Name: C Flat

Group Leader: Job Lipat

Group Members:

Charmaine Eunice Rabano

Madeline Isabel Galang

Ryka Santos

Mark Anthony Mamauag

1. General Description:

C Flat (Stylized as“C*b*”) is an interpreted and statically typed programming language that is heavily inspired by C and Python. The objective of the language is to remove the complexity of C while preserving its static syntax. C Flat achieves this by abstracting features such as pointers, boilerplate code etc.. It also does this by integrating some language features of Python such as conditional statements and logic operators to make the language more readable.

2. Basic symbols:

Identifiers	
`<identifier>`	Represents values. Must start with letters followed by alphanumeric characters. Must not equal to other keywords Regex:`[a-zA-Z][a-zA-Z0-9]*`

Data Types	
int	Integer. Numbers that don’t have a decimal part
string	String. Combination of characters
bool	Boolean type. Is either true or false

Literals	
<integer_literal>	Actual integer values. Regex: (-)?[0-9]+
<string_literal>	Actual string values. Regex: ".*"
<boolean_literal>	Actual boolean values. Regex: True|False

Comments	
//	Single line comment
/* */	Multi line comment

I/O	
print	Output. A function that outputs a string of characters to the console
input	Input. A function that gets a string from the user

Delimiters	
,	Separates parameters and arguments
( )	Denotes parameters or arguments of a function Groups logical and arithmetic expressions
;	Terminates a simple statement
{ }	Denotes code block

Part of Conditional Statements	
if	Executes block if condition is true
elif	Executes block if previous if or elif is false
else	Executes block if all elif are false

Function	
void	Used as a function return type which specifies that the function does not return a value
return	Ends the execution of a function and send the function's result back to the caller

Loops	
while	Executes block until condition is false
for	Repeats a block for a known number of times

Assignment	
set	Used to assign a value to an identifier

Operators		
Assignment Operator	=	Sets the value of the identifier to the value of the given expression
Logical operators	and	A binary operator that evaluates to true if both expressions are true. Otherwise, it evaluates to false.
	or	A binary operator that evaluates to true if at least one of the expressions is true. Otherwise, it evaluates to false.
	not	A unary operator that evaluates to true if the given expression is false. Otherwise, it evaluates to false.
Arithmetic Operator	+	A binary operator that **adds** two integers together
	-	A binary operator that **subtracts** the integer in the left hand side by the integer in the right hand side
	*	A binary operator that **multiplies** two integers together
	/	A binary operator that **divides** the integer in the left hand side by the right hand side. The expression will evaluate into an integer and the decimal part will be discarded
Comparison/ Relational Operator	==	A binary operator that evaluates to true if the two expressions are **equal**. Otherwise, false.

	!=	A binary operator that evaluates to true if the two expressions are **not equal**. Otherwise, false.
	>	A binary operator that evaluates to true if the right hand side is **greater than** the left hand side. Otherwise, false.
	<	A binary operator that evaluates to true if the right hand side is **less than** the left hand side. Otherwise, false.
	>=	A binary operator that evaluates to true if the right hand side is **greater than or equal to** the left hand side. Otherwise, false.
	<=	A binary operator that evaluates to true if the right hand side is **less than or equal** to the left hand side. Otherwise, false.

3. Grammar

<program>	<block>
<block>	<statement>*

Variable Definition Grammar	
<declaration>	<data_type> <identifier> "=" <expression>
<assignment>	"set" <identifier> "=" <expression>

Expression Grammar	
<expression>	<literal> | <identifier> | <arithmetic_expression> | <logical_expression> | <relational_expression> | <function_call>
<arithmetic_expression>	*!!simplified for readability!!* <arithmetic_expression> <arithmetic_operator> <arithmetic_expression>

<logical_expression>	*!!simplified for readability!!* <logical_expression> <logical_operator> <logical_expression>
<relational_expression>	<logical_expression> <relational_operator> <logical_expression>
<function_call>	<identifier> "(" <argument_list> ")"
<argument_list>	(<expression> ("," <expression>)*)?

**for readability,* <arithmetic_expression> *and* <logical_expression> *are simplified. Full BNF grammar is defined below.*

Statement Grammar	
<statement>	<expression> ";" | <declaration> ";" | <assignment> ";" | <loop_statement> | <selection_statement> | <function_declaration>
<selection_statement>	"if" "(" <expression> ")" "{" <block> "}" ("elif" "(" <expression> ")" "{" <block> "}")* ("else" "{" <statement> "}")?
<loop_statement>	<for_loop> | <while_loop>
<for_loop>	"for" "(" <declaration> ";" <expression> ";" <assignment> ")" "{" <block> "}"
<while_loop>	"while" "(" <expression> ")" "{" <block> "}"

Procedure Grammar	
<function_declaration>	<return_type> <identifier> "(" <parameter_list> ")" "{" <block> <return>?"}"
<parameter_list>	(<parameter> ("," <parameter>)*)?
<parameter>	<data_type> <identifier>
<return>	"return" <expression> ";"
<return_type>	<data_type> | "void"

4. **Rules/Restriction**

Lexical Errors:

Error	Wrong symbol
Message	Token not recognized.
Example	int num =@= 5; print["Hello world"];

Error	Identifier not following pattern
Message	Identifier not accepted. Identifiers should consist of English letters and numbers, and should start with an English letter.
Example	int $num_ = 1;

Error	Not closing string
Message	Invalid string. String should be closed.
Example	string x = "Hello;

Error	Not closing multiline comment
Message	Invalid multiline comment. Multiline comments should be closed.
Example	/*multiline comment

Syntax Errors:

Error	Missing semicolon
Message	Semicolon ';' expected.
Example	int a = 1 + 1

Error	Missing coma
Message	Coma ‘,’ expected.
Example	int functionName (<parameter1> <parameter2>)

Error	Missing braces
Message	Open brace '{' expected, Close brace '}' expected.
Example	if (true) {

Error	Missing parenthesis
Message	Open parenthesis '(' expected, Close parenthesis ')' expected.
Example	print(

Error	Incorrect placement of binary operator
Message	Binary operator should be between two expressions.
Example	int num = * 5;

Error	Incorrect placement of unary operator
Message	Unary operators should be in front of an expression.
Example	bool varName = not;

Error	Incorrect “if”, “elif”, “else” order.
Message	"if" should come before "else", "if" should come before "elif", "elif" should come before "else"
Example	else { <statement> } if (true) { <statement> }

Error	Incorrect/incomplete grammar.
Message	Expression expected., Declaration expected., Data type expected., Identifier expected.
Example	for ( ) { <statement> } string = “Hello world!”;

Error	Keyword as an identifier
Message	Keyword can't be used as identifier.
Example	int for = 1;

Semantic Errors

Declaration Grammar

Error	Variable declaration type error
Message	Data type and expression must match.
Example	int varName = "1"; string varName = 1;

Error	Variable assignment type error
Message	variable data type and expression must match.
Example	int varName = 1; varName = "1";

Error	Declaration of identifiers cannot be declared more than once
Message	varName cannot be declared more than once
Example	int varName = 1; Int varName = 2;

Expression Grammar

Error	undeclared variable
Message	“a” is not defined
Example	1 + a;

Error	Arithmetic expression type error
Message	Arithmetic expression expects integer type
Example	1 + “abc”;

Error	logical expression type error
Message	Logical expression expects boolean type
Example	1 or false;

Error	relational expression type mismatch
Message	Expressions must have the same data type
Example	“” > 2

Error	Invalid argument count
Message	Argument list should contain the same number of arguments present in the declared function.

Example	Int add(int x, int y){
	return x + y;
	}
	
	Int a = add(5,6,7);

Error	Invalid argument type
Message	Argument type must match parameter type

Example	Int add(int x, int y){
	return x + y;
	}
	
	Int a = add(5, "");</br>

Statement Grammar

Error	Selection statement type error
Message	Expressions must evaluate to a boolean value
Example	if (1 + 2){ }

Error	For loop type error
Message	Expression must evaluate to a boolean value
Example	for(int i = 0; 1 + 1; i = i + 1){ }

Error	While loop type error
Message	Expression must evaluate to a boolean value
Example	while(1){ }

Procedural Grammar

Error	Variable out of scope
Message	Variable not found

Example	Int a = 5;
	int add(int x, int y){
	return x + y + a;
	}

Error	missing function return statements
Message	Value returning function is expected to return a value

Example	Int a = 5;
	int add1(int x, int y){
	x + y;
	}
	
	int add2(int x, int y){
	if(x < y) { return x + y;}
	}

Error	function return type error
Message	“message” function expects to return a string

Example	string message(string a){
	print(a);
	return 1;
	}

Error	void function with return value
Message	“message” function does not expect a return value

Example	void message(string a){
	print(a);
	return 1;
	}

5. Test Cases

a. Valid

Test PRINT statements

```
print("Hello world!");
print("Have a good day!");
```

Test PRINT before INPUT, INPUT and DECLARATION

```
print("Enter a word:");
string x = input();

print("The word you entered:");
print(x);
```

Test arithmetic operators, ASSIGNMENT and NEGATE

```
int a = 10 / 2 * 3;
set a = - a;
print(a);
```

Test FOR loop statement and LESS operator

```
for (int i = 0; i < 10; set i = i + 1) {
    print(i);
}
```

Test WHILE loop statement and MORE_EQUAL operator

```
int i = 5;
while(i >= 0) {
    print(i);
    set i = i - 1;
}
```

Test IF, ELIF, ELSE statements and DOUBLE_EQUAL operator

```
print("Enter a letter:");
string l = input();

if (l == "a") {
    print("You entered a");
}
elif (l == "b") {
    print("You entered b");
}
else {
    print("You didn't enter a or b");
}
```

Test NOT and MORE operators

```
print(not (5 > 5));
```

Test OR, NOT_EQUAL and LESS_EQUAL operators

```
print((2 != 2) or (4 <= 8));
```

Test AND operator and booleans

```
print(True and False);
```

Test arithmetic and logical operators precedence

```
int x = 2 + 4 * 6 - (- 8 / 4);
bool y = 10 == 10 or not (2 < 5) and False;

print(x);
print(y);
```

b. Invalid

i. Lexical Analyzer

ERROR TYPE: Token Error. There's no token that has dollar sign

```
//single line comment try
int $num = 1;
```

ERROR TYPE: Token Error. There's no token that has underscore.

```
/*multi line
comment try*/
int _num = 1;
```

ERROR TYPE: Token Error. There's no token that has a period.

```
/*multi line
comment try*/
print("hello\nworld").
```

ERROR TYPE: Token Error. Multiline comment didn't close.

```
/*multi line
comment try
```

ERROR TYPE: Token Error. There's no token that uses exclamation point aside from "!=".

```
/*multi line
comment try*/
if (!haha == 5){
    print("hello");
}
```

ERROR TYPE: Token Error. String didn't close. `"stri`

ii. Syntax Analyzer

Semicolon
No semicolon after declaration. `int a = 1`

No semicolon after assignment statement
`int a = 1;`
`set a = 1`

No semicolon after print statement `print("hello")`
No semicolon after declaration statement with expression `string a = input()`
Declaration
Missing expression after equal symbol `int a = ;`
Missing identifier `int = 1;`
Missing data type `a = 1;`
Assignment
Missing expression after equal symbol `set a = ;`
Expressions
Missing right hand side expression `int a = 1 +;`
Missing both left and right hand expressions `int a = > ;`

Missing left hand side in expression

```
int a = * 1;
```

Missing expression after unary operator

```
bool a = 1 not ;
```

Conditional

Wrong if keyword

```
i(){

}
```

No condition

```
if(){

}
```

Missing opening curly braces

```
if(True)

}
```

Missing closing curly braces

```
if(True){
```

Missing both curly braces

```
if(True)
```

Missing expression in elif

```
if(True){
    print("Hello");
}
elif(){

}
```

Missing closing curly braces

```
if(True){
    print("Hello");
}
elif(True){
```

Missing opening curly braces

```
if(True){
    print("Hello");
}
```

```
elif(True)

}
```

Missing closing parenthesis

```
if(True){
    print("Hello");
}
elif(True{

}
```

Missing opening parenthesis

```
if(True){
    print("Hello");
}
elif True){

}
```

Missing closing curly braces

```
if(True){
    print("Hello");
}
elif(True){

}
else{
```

Missing opening curly braces

```
if(True){
    print("Hello");
}
elif(True){

}
else

}
```

Wrong keyword for else

```
if(True){
    print("Hello");
}
elif(True){
```

```
}
el{

}
```

For loops

Wrong keyword for for loop

```
fr(int i = 0; i < 10; set i = i + 1){

}
```

Missing opening curly braces

```
for(int i = 0; i < 10; set i = i + 1)

}
```

Declaration instead of assignment in the third parameter

```
for(int i = 0; i < 10; int j = 1){

}
```

Assignment instead of declaration in the first parameter

```
int i = 0;
for(set i = 0; i < 10; set i = i + 1){

}
```

No third parameter

```
for(int i = 0; i < 10;){

}
```

No second parameter

```
for(int i = 0; ; set i = i + 1){

}
```

No first parameter

```
for(; i < 10; set i = i + 1){

}
```

Missing closing curly braces

```
for(int i = 0; i < 10; set i = i + 1){
```

While loops

Missing opening curly braces
`int i = 0;`
`while(i < 5)`
`    set i = i + 1;`
`}`

Missing closing curly braces
`int i = 0;`
`while(i < 5){`
`    set i = i + 1;`

Missing close parenthesis
`int i = 0;`
`while(i < 5{`
`    set i = i + 1;`
`}`

Missing open parenthesis
`int i = 0;`
`whilei < 5){`
`    set i = i + 1;`
`}`

Missing Parentheses
`int i = 0;`
`while i < 5 {`
`    set i = i + 1;`
`}`

Assignment instead of expression in condition
`int i = 0;`
`while(set i = 5){`
`    set i = i + 1;`
`}`

iii. Semantic Analyzer

ERROR TYPE: Mismatched Data Type `int num1 = "fifteen";`
ERROR TYPE: Mismatched Data Type `string stringVar = 1;`

ERROR TYPE: Identifier {x} cannot be declared more than once

```
int x = 10; int x = 20;
```

ERROR TYPE: Identifier {x} cannot be declared more than once

```
string test = "case"; string test = "case";
```

ERROR TYPE: Identifier {x} cannot be declared more than once

```
string test = "case"; string test = "file";
```

ERROR TYPE: Variable {total} is not defined

```
int pcs = 5; int qty = pcs * total;
```

ERROR TYPE: Arithmetic expression expects integer type

```
string pcs = "hello"; int total = 50; int qty = 20 * pcs;
```

ERROR TYPE: Arithmetic expression expects integer type

```
string x = 1 + 2;
```

ERROR TYPE: Arithmetic expression expects integer type

```
string x = 1 - 2;
```

ERROR TYPE: Arithmetic expression expects integer type

```
string x = 1 * 2;
```

ERROR TYPE: Arithmetic expression expects integer type

```
string x = 1 / 2;
```

ERROR TYPE: Arithmetic expression expects integer type

```
string pcs = "hello"; int total = 50; int qty = 20 * pcs;
```

ERROR TYPE: Arithmetic expression expects integer type

```
int num1 = 1;
int num2 = 5;
string x = num1 * num2;
```

ERROR TYPE: Logical expression expects boolean type

```
bool varName = True or 1;
```

ERROR TYPE: Logical expression expects boolean type

```
bool varName = 1 and True;
```

ERROR TYPE: Logical expression expects boolean type

```
bool varName = not 1;
```

ERROR TYPE: Logical expression expects boolean type

```
bool varName = (1 + 1) and True;
```

ERROR TYPE: Relational operation expects same data type

```
int x = 10;
if (x > "hello"){
```

```
    print(x);
}
else{
    print("hello");
}
```

ERROR TYPE: Relational operation expects same data type

```
string one = "good";
string two = "bad";
if (one == 2){
    print("nice");
}
else{
    print("meh");
}
```

ERROR TYPE: Expressions must evaluate to a boolean value

```
int a = 15;
int b = 5;
set b = 10;

if (a + b){
    print(a);
}
```

ERROR TYPE: Expressions must evaluate to a boolean value

```
for (int i = 0; 1 + 1; set i = i + 1)
{
  print (i);
}
```

ERROR TYPE: Expressions must evaluate to a boolean value

```
int i = 0;
while (1)
{
  print(i);
}
```

ERROR TYPE: Invalid type when declaring boolean variable

```
bool a = 1;
```

ERROR TYPE: Invalid left hand side in arithmetic operators

```
int a = False + 1;
```

ERROR TYPE: Equality operator data types not equal

```
bool a = 1 == False;
```

ERROR TYPE: Using integer negation on boolean type

```
int a = -False;
```

References:

https://www.lua.org/manual/5.3/manual.html [Lua - very simple syntax - scroll to very end]

http://www.nongnu.org/hcb/ [C++ Grammar Specification]

https://cs.wmich.edu/~gupta/teaching/cs4850/sumII06/The%20syntax%20of%20C%20in%20Backus-Naur%20form.htm [C Grammar Specification]

https://www.cs.unc.edu/~plaisted/comp455/Algol60.pdf [Algol60 Grammar Specification]

https://docs.python.org/3/reference/grammar.html [Python Grammar Specification]

BNF Grammar

```
Note: ? means 0 or 1, * means 0 or N, + means 1 or more. Some parts are in Regex

<program> ::= <block>
<block> ::= <statement>*

<data_type> ::= "int" | "string" | "bool"
<identifier> ::= [a-zA-Z][a-zA-Z0-9]*
<literal> ::= <string_literal> | <integer_literal> | <boolean_literal>
<string_literal> ::= "\"" .* "\""
<integer_literal> ::= ("-")? [0-9]+
<boolean_literal> ::= "True" | "False"

<statement> ::=
<declaration> ";"
| <assignment> ";"
| <loop_statement>
| <selection_statement>
| <function_declaration>

<selection_statement> ::=
"if" "(" <expression> ")" "{" <block> "}"
("elif" "(" <expression> ")" "{" <block> "}")*
("else" "{" <statement> "}")?

<loop_statement> ::=  <for_loop> | <while_loop>
<for_loop> ::= "for" "(" <declaration> ";" <expression> ";" <expression> ")" "{"
<block> "}"
<while_loop> ::= "while" "(" <expression> ")" "{" <block> "}"

<declaration> ::= <data_type> <identifier> "=" <expression>
<assignment> ::= "set" <identifier> "=" <expression>
```

```
<expression> ::=
<literal>
| <identifier>
| <arithmetic_expression>
| <logical_expression>
| <relational_expression>
| <function_call>

<arithmetic_expression> ::= <arithmetic_expression> "+" <term> |
<arithmetic_expression> "-" <term> | <term>
<term> ::= <term> "*" <primary> | <term> "/" <primary> | <primary>
<primary> ::= "(" <arithmetic_expression> ")" | <integer_literal>

<logical_expression> ::= <logical_factor> | <logical_expression> "or" <logical_factor>
<logical_factor> ::= <logical_primary> | <logical_factor> "and" <logical_primary>
<logical_secondary> ::= "not" <logical_primary> | <logical_primary>
<logical_primary> ::= "(" <logical_expression> ")" | <boolean_literal>

<relational_expression> ::= <logical_expression> <relational_operator>
<logical_expression>

<function_call> ::= <identifier> "(" <parameter_list> ")"
<argument_list> ::= (<expression> ("," <expression>)*)?


<function_declaration> ::= <data_type> <identifier> "(" <parameter_list> ")" "{"
<block> "}"
<parameter_list> ::= (<parameter> ("," <parameter>)*)?
<parameter> ::= <data_type> <identifier>
```

Sample Program

```
// C Flat programming language
// Cf
// A C and python inspired language
// A more succinct and user friendly version of C
// removes pointers
// removes boilerplate code

// Comments in C Flat
// single line comment
/*multi line
```

```
comment*/

// data types
// string -> ""
// int -> 0
// boolean -> True, False

// identifiers
// [a-zA-z][a-zA-z0-9]*
// a1s1s11ss11ss1

// variables
int a = 0;

// I/O
print("Hello world");
string s = input("Enter your name");

// Operations
// Arithmetic
// +
// -
// *
// -
int b = 1 + 1;
set b = a + 1;
int c = a + b;

// logical
// and
// or
// not

// relational
// ==
// >
// <
// >=
// <=

// assignment
// =


// control structures
// conditional
```

```
if(True){

}

if(True){

} else {

}

if(True){

} elif(True) {

}

if(True) {

}
elif(True) {

} else {

}


// looping
for(int i = 0; i < 10; set i= i + 1){

}

// functions
int function(){
    return 0;
}

void function(){

}

int addThree(int a){
    return a + 3;
}
```