BuckCal

Final Report

Team

Ahmad Maruf (aim2122)
Lan Yang (ly2331)
Lingyuan He (lh2710)
Meng Wang (mw2972)
Prachi Shukla (ps2829)

Programming Language and Translator
COMS W4115 (Fall 2014)
Prof. Stephen Edwards
Date: December 17, 2014

Table of Contents

Chapter 1

Introduction

"There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."
– C.A.R. Hoare (British Computer Scientist, Winner of the 1980 Turing Award)

BuckCal is a matrix manipulation language. It has full support for mathematical matrix operations and is optimized for spreadsheet calculations. With enhanced matrix operations and data type support, programmers can easily make budget, record and calculate expenses. Programs written with BuckCal may include employee payroll calculator, statistical computation, and estimation of budget.

1.1. Motivation

Calculating expenses is amongst the most common daily tasks. For example, keeping track of our daily or weekly expenses or how much money we owe to others. We felt the need to design a language that will record all data in a matrix in order to calculate daily expenses on various commodities and estimate savings. A language that can allocate the bills to members of a matrix, so that each member gets a clear idea of how much he or she owes to others.

1.2. Why BuckCal?

Someone may suggest numeric computing software such as MATLAB and visual spreadsheet processing application such as Microsoft Excel as alternatives. However, while MATLAB is not convenient enough for simple daily use, Microsoft Excel is not quite designed for straightforward programming. To solve these problems, BuckCal combines the advantages of both MATLAB and MS Excel by offering powerful computation and ease of use. BuckCal is a scripting language that combines element from Python, Bash etc.; it is easy to learn for any one familiar with spreadsheet such as Excel and Google Spreadsheet, and is of course easy to use for people with some programming experience. Therefore, to code in BuckCal, no serious programming skills is required. It has simple built-in matrix support and functionality, and therefore to program in BuckCal, one doesn't need to learn MATLAB beforehand, or know C++ just for its library. Furthermore, BuckCal library is easily expandable with convenient *import* option (more on that later). In short, our project aims to introduce a simple, comprehensible, and easily programmable language to analyze the money we spend, but we surely do not intend to prevent anyone from learning MATLAB, MS Excel, C++, or any other tools for that matter!

Chapter 2

Language Tutorial

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."
~ Martin Fowler (Author and speaker on software development)

2.1 Program Execution

To compile (but not run) a .bc program, simply use the compilation shell script with your .bc file as the only argument.
./compile.sh <main buckcal source file> <output executable name>

To run this program, use the run script. This script calls the compilation script first, so no compilation is required before using it.
./<output executable name>

Note: Only the main executable BuckCal file should be provided to the script, imported files should present, but their names are not required for **compile.sh**.
The built-in library **buckcal_lib.bc** has to be properly imported (using **import**; see section 3.10) when used, but it does not have to be copied by user, the script will do that.

2.2. Variables

All variables must be declared with its data type before used. For example,

```
int a : 1 + 2;
double d ;
string c : 'This is a string';
bool b;
```

If variables are declared without initial value, their default values are as follows.

```
int a; # a = 0
double d; # d = 0.0
string s; # s = ' '
bool b; # b = false
```

More on variables are discussed in sections 3.4.-3.6.

2.3. Matrices

To create a non-empty matrix m, **mat** literal is needed. For example,

```
int mat m: {1, 2; 4, 5};
```

Matrix variable declared without initial value is empty. Also, sub-matrix expression is for accessing a single element of a matrix. The generic form is:

matname [row, col];

row and *col* should be positive integers, beginning with 1. For matrix operations, BuckCal library functions, explained in section 8.1.1., should be called. More on matrix operations is discussed in section 3.11.

2.4. Control Flow

BuckCal provides control flow statements, which allow the programmer to change the CPU's path through the program. There are quite a few different types of control flow statements, so we will cover them briefly here, and then in more detail throughout the section 3.8.

BuckCal supports 'if/elif/else' statements:

```
if (mybool1 = true) then
        disp 'inside the if';
elif (mybool2 = true) then
        disp 'inside the elif';
else
        disp 'inside the else';
fi
```

BuckCal supports 'for loops':

```
int r;
for r in {1, 2; 3, 4} do
        disp r: r * 4;
rof
```

Output:

4
8
12
16

```
int i : 5;
string text : 'The number is:';
for i < 8  do
       disp text;
       disp i;
```

Output:

The number is:
5
The number is:
6

i : i + 1; **rof**	The number is: 7

2.5. Functions

# function definition **def** hello() **do** **disp** 'hello'; **fed**	**Output:** hello

For more on functions, see section 3.9.

2.6. Printing to stdout

The keyword **disp** prints any literal and expression with a non-void return value.

```
disp 'hello world!';   # prints to stdout: hello world

string expr = 'PLT was so much fun! Those RED BULLS helped a lot!';

disp expr;     # prints to stdout: PLT was so much fun! Those RED BULLS helped a lot!
```

disp will print the literals and value of expression to stdout. For more discussions on **disp**, see section 3.8.4.

Chapter 3

Language Reference Manual

"Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live."
– Martin Golding

3.1. Lexical Component

3.1.1. Keywords

Keywords are reserved for language processing, and they cannot be used for identifier or other purposes.
All keywords:

if, then, else, elif, fi, for, in, do, rof, disp, **break, continue, def, fed, return,** **not, and, or, int, double, bool, string,** **int mat, double mat, string mat,** **true, false, import**

3.1.2. Identifiers

An identifier is used to name a variable or function, it could be any combination of lowercase letters (a to z), number (0-9) or underscore(_), except that the first letter must be a lowercase character (a to z). An identifier cannot be an exact match with one of the reserved keywords.

Examples of valid identifiers:	Examples of invalid identifiers:
i matrix01 food_day2	Var (uppercase letter) number (number as the first letter) an int (with space) *str (invalid character)

3.1.3. Literals

A literal is used to express a constant value, which can be of type: **int**, **double**, **string**, **int mat, double mat**, and **string mat**. More details about data types are in section 3.2.
Examples:

1	**int**
1.0	**double**
true	**bool**
'h'	**string**
{1,2,3,4}	**int** matrix
{'h', 'a'}	**string** matrix

A string literal must be quoted by single quote. For **mat** literal, columns are separated with a comma, and rows are separated with a semicolon. All rows must have the same number of elements. An empty **mat** (if initialized with '{ }') literal is illegal.

3.1.4. Separators

Semicolon - ';'	Separated variable declarations and statements.
Comma - ','	Separates arguments in a function argument list, also separates two elements in same row in a matrix literal.
Curly bracket - '{ }'	Used to wrap a matrix literal.
Parenthesis - '( )'	Used to modify operator precedence, and to wrap the argument list of a function call.

3.1.5. Operators

Plus '+'	Mathematical addition for numeric values, concatenation for strings.
Minus '-'	Mathematical subtraction for numeric values.
Multiply '*'	Mathematical multiply for numeric values.
Divide '/'	Mathematical division for numeric values.
Assign ':'	Used in assignment and variable initialization.
Logical equal '=' and Logical unequal '!='	Used only for logical comparison, return a **bool.**
'<' '>='	Tests if the left operand is less than right operand. Returns the opposite.
'>' '>='	Tests if the left operand is greater than right operand. Returns the opposite.
Index - '[ ]'	Used in matrix subscripting.
not	Logical NOT. Can be applied to **bool** expression only.
and	Logical AND. Can be applied to **bool** expression only.
or	Logical OR. Can be applied to **bool** expression only.
string	A string is compared by dictionary order.
bool	For Boolean, true id greater than false.

The operators (+ - * /) applies the same for any of type: **int, double, int mat, double mat**. For example, operator + applies not only for numeric, but also between numeric value and matrix (scalar addition) or between matrix (vector addition).

Specifically, below are the mathematical operators (+ - * /) and examples of their usage.
num : (**int** | **double**);

	+	-	*	/
num op num	1+2.1=3.1	1-2.1=-1.1	1.0*2.1=2.1	2.1/1.0=2.1
string op string	's'+'u'->'su'	N/A	N/A	N/A
matrix op num	{1, 2; 3, 4} + 1 = {2, 3; 4, 5}	{1, 2; 3, 4} - 1 = {0, 1; 2, 3}	{1, 2; 3, 4} * 1.2 = {1.2, 2.4; 3.6, 4.8}	{1, 2; 3, 4} / 2.0 = {1.5, 1.0; 1.5, 2.0}
matrix op matrix	{1, 2; 3, 4} + {1, 2; 3, 4} = {2, 4; 6, 8}	{1, 2; 3, 4} - {1, 2; 3, 4} = {0, 0; 0, 0}	{1, 2; 3, 4} * {1, 2; 3, 4} = {1, 4; 9, 16}	{1, 2; 3, 4} / {1, 2; 3, 4} = {1, 1; 1, 1}

matrix : (**int** mat | **double** mat);
Details on operators and operation can be found in section 3.3. ('Expression').

3.1.6. Newlines, Whitespaces and Tabs

Newlines ('\n'), whitespaces (' '), and tabs ('\t') are used to split lexical components. Using one or more or any combination of them will make no difference.

3.2. Data Types

In BuckCal, these are primitive data types: number (integer and floating point number), string, and boolean. Data with same primitive data type can be composed into a matrix.

3.2.1. Primitive Data Types

int

The **int** data type can hold 32-bit signed integer values.

double

The **double** type is 64-bit IEEE double precision floating point number.

string
A **string** represents a collection of characters. Strings are constant and immutable; their values cannot be changed after they are created. And follow escape characters are supported: '\n' '\t' '\".

boolean
The boolean data type (**bool**) has only two possible values: **true** and **false**, but its size is undefined.

3.2.2. Matrix
A matrix (***mat***) is a data collection type, which can store data of a single primitive data type except boolean. That means a matrix can be an **int** matrix, **double** matrix or **string** matrix.

3.2.3. Data Types Conversion
Built-in functions are provided support data type conversion. The function naming convention is:

def newtype newtype_of_oldtype(oldtype x);

The conversions we support are:

int ←→ double
(int | double) ←→ string
(int mat | double mat) ←→ string mat

Note that **int** (**mat**) and **double** (**mat**) can be converted implicitly, which means they can be used interchangeably. So there are no additional **int→double** conversion functions.

3.3. Expressions
Expressions are made up of variable, literal, sub-matrix and function call. These components are combined together by operators and "( )". Because expressions have a return value, an expression itself can be embedded in a bigger expression. The only exception is function call that returns "**void**". More details about sub-matrix expressions can be found in section 3.11.2.

3.4. Variable Declaration
All variables must be declared with its data type before used. The initial value is optional; but if there is one, it must be an expression resulting in the same type with variable. Grammar:
datatype identifier [: initial value]

Samples are as follows:

```
int a : 1 + 2;
double d ;
string c : 'This is a string';
bool b;
```

If variables are declared without initial value, their default values are as follows.

```
int a;              # a = 0
double d;           # d = 0.0
string s;           # s = ' '
bool b;             # b = false
```

To create a non-empty matrix, mat literal is needed. Example:

```
mat m: {1, 2; 4, 5};
```

Matrix variable declared without initial value is empty.

3.5. Variable Scope

There are two levels of scope: top and function. A *top variable* is a variable with top scope, whereas a *function variable* with function scope.

A variable defined within a function has a function scope. It can only be referenced within the function where it is defined. Top variables are defined out of function. Scopes are isolated from each other: a variable defined in top variable cannot be referenced within a function. If a function variable has the same name as a top variable, it is treated as a different variable and has no connect with the top one.

3.6. Variable Assignment

The assignment operator ":" stores the value of its right operand in the variable specified by its left operand. The left operand (commonly referred to as the "left value") can only be a single variable or sub matrix expression.

The left and right operand should be of the same type. The only exception is that integer and double can be assigned to each other. Only the integer part of a double will be assigned to an **int.**

Example:

```
double b : 1.2;
int a;
a : b; # a = 1
```

3.7. Operator Precedence

Rules of precedence ensure when dealing with multiple operators, codes can be concise and simple while not having ambiguity. A simple example of precedence is a: b + c * d. This expression means that the result of c multiplying with d is added to b, and then the addition result is assigned to a.

In BuckCal, most operators are left-associated, some special cases would be stated later. The orders of highest precedence to lowest precedence follow the list below. Note that two or more operators may have same precedence.

- Parenthesis '( )'
- Function calls, matrix subscripting
- Unary negative
- Multiplication, division expressions
- Addition and subtraction expressions
- Greater-than, less-than, greater-than-or-equal-to and Less-than-or-equal-to expressions
- Equal-to and not-equal-to expressions
- Logical NOT expressions
- Logical AND expressions
- Logical OR expressions
- Assignment expressions

3.8. Statement

The simplest statement is an expression with a semicolon at the end.

3.8.1.Conditional Statement

The **if-then-else-fi** statement is used to conditionally execute part of the program, based on the truth-value of a given expression. Here is the general form of an **if-else-fi** statement:

```
if bool-expr then
      statement1;
else
      statement2;
fi
```

If bool-expr evaluates to **true**, then only statement1 is executed. On the other hand, if bool-expr evaluates to **false**, then only statement2 is executed. The **else** clause is optional.

The **if-then-elif-then-else-fi** statement is used to cascade the conditional execution of the program. Here is the general form of an **if-elif-else-fi** statement:

```
if bool-expr1 then
      statement1;
elif bool-expr2 then
      statement2;
elif bool-expr3 then
      statement3;
else
      statement4;
fi
```

Just like in **if-then-else-fi**, the **else** clause is optional here.

3.8.2. Loop Statement

3.8.2.1. Counting Loop

The counting loop statement iterates over a mat row by row. The iterate variable must be pre-defined. Here is an example:

```
int r;
for r in {1, 2; 3, 4} do
    #do something;
rof
```

In the above example, r traverses through 1, 2, 3, 4.

3.8.2.2. Conditional Loop

The conditional loop iterate until the given condition becomes false. Here is the general form:

```
for bool-expr do
   # do something;
rof
```

3.8.3. *break* and *continue* Statements

The **break and continue** statements must be used within for loop.
The **break** statement terminates a for loop. The **continue** statement terminates current iteration and begins the next iteration.

3.8.4. *disp* Statement

The keyword **disp** can print any literal and expression with a non-void return value. The format is—

disp expr;

This will print the value of expression to stdout, in a predefined pretty style. Note that **bool** value is printed as 1 (for **true**) and 0 (for **false**)

3.9. Functions

BuckCal provides some built-in functions (See section 8.1.1.1.). Users can also define their owns.

3.9.1.Function Declarations

A function declaration is to specify a function's return value type, the name of function, and a list of types of arguments. The general form:

def [*type*] identifier(argument-list);

The declaration begins with **def** keyword, and the return value type. If the function returns void, then the *type* can be omitted. What follows **def** is the function name, which should be a valid identifier. The list of arguments can go between a pair of parentheses. Note that the names of formal variables are optional. In fact, the names (if any) are ignored, and only types take effect. Finally it ends with a semicolon (';'). Function declaration can only be in the top level.

3.9.2 Function Definitions

A function definition begins with a declaration-similar part, which specifies the name of the function, the argument list, and its return type. If an argument has no name, it cannot be referred to in function body.

The general form:

```
def [type] identifier(argument-list) do
      function-body;
fed
```

Keywords **do** and **def** wrap the function body. In the function body is the declarations of local variables, and a list of statements to be executed when function is called. Note that all variable declarations must appear before any statements. For a function with non-void return type, there must be a return statement in the statements; and expression returned must have the same type with function return type. A function definition cannot appear in another function's definition. That means it only exists in the top-level code. Recursion is also supported in BuckCal.

3.9.3. Function Overloading

As is in C++, BuckCal allow functions have the same name, as long as the number or types of arguments are not identical. For example, some library functions are overloaded:

```
int cols(int mat);
int cols(double mat);
int cols(string mat);
```

The above three functions have the same name, same number of argument, but type of arguments are different. Thus the overloading is valid.
Note that overloading by return type is not acceptable. That means user cannot define two functions almost identical except in return type.

3.9.4. Calling Functions

BuckCal built-in functions can be called anywhere, anytime in a program. User-defined function must be defined before called. Arguments are passed by value. The only exceptions are in some built-in functions, which accept arguments by reference. The arguments are evaluated before function call, and the order of argument evaluation is unspecified.

3.10. *import* Instruction

The keyword **import** will make function definitions in another .bc file available for current .bc file. The **import** instruction should be placed in the very beginning of a BuckCal source file. The generic form is:

import *filename*

The *filename* is a string literal, containing the relative path of .bc file from current source file directory. Note that a semicolon is not needed after <*filename*>.

3.11. Matrix Operations

Because of the complexity of matrix operation, here is a chapter specially for matrix.

3.11.1. Type of Matrix

According to the type of its elements, a matrix should be one of the three subtypes: **int mat**, **double mat**, and **string mat**.

3.11.2.Sub-Matrix Expressions

Sub-matrix expression is for accessing a single element of a matrix. The generic form is:

matname [*row*, *col*];

row and *col* should be positive integer, beginning with 1. The return type of a sub matrix expression is the same with element type, and it's both readable and writeable: assignment into a sub matrix expression changes the corresponding matrix.

3.11.2. Row/Column Name

By default, a matrix has row and column named. The names can be change by built-in functions. When **disp** a matrix, the row/column names are also printed.

3.12. Sample Programs

3.12.1. “Hello World” Style

```
# function definition
def hello() do
        disp 'hello';
fed

# top level variable declaration
int a : 5;
int i : 0;
int x : 0;
int mat v ;
double b : 0.0;
string endstr : 'End' ;

# top level code
hello();
for i do
        if i <= 5 then
        v : colcat(v, {i}); # call built-in function
        i : i + 1;
        else
        break;
        fi
orf
for x in v do
        if x <= 2 then
        b : b + x*x;
        elif x <= 4 then
        b : b + x;
        else
        b : b + x/2.0;
        fi
rof
disp endstr;
```

3.12.2. Matrix Operation

```
# import library
import ‘buckcal_lib.bc’
import 'imp.bc'

def double mat addrow(double mat a, double mat b, string mat s) do
    double mat tmp : b;
    rowname(tmp, s);
    tmp: rowcat(a, tmp);
    return tmp;
fed

# declare top variable
double mat budget;
double mat subdget;
double mat tmp;

# initialize
budget: {1+1, 3.3};
colname(budget, {'Food', 'Price'});
rowname(budget, {'John'});
# add one column and naming
tmp : {0};
colname(tmp, {'Paper'});
budget: colcat(budget, tmp);
# add one row with naming
budget: addrow(budget, {250*2, 0, 5.10}, {'Tom'});
# add sum row
budget: addrow(budget, sum_row(budget), {'sum'});
# display
disp budget;
```

==========

Note: The output of disp statement is as follows:

```
        Food    Price    Paper
John    [2,    3.3,    0]
Tom     [500,    0,    5.1]
sum     [502,    3.3,    5.1]
```

Chapter 4

Project Plan

"Those who plan do better than those who do not plan even though they rarely stick to their plan." – Winston Churchill

"Everyone has a plan: until they get punched in the face." – Mike Tyson

For our project, without careful planning and organization we wouldn't have seen the light. We tried our best to implement a number of simple project management systems to ensure success. This section outlines the techniques we used to make BuckCal come true!

4.1 Project Processes

4.1.1 Planning

We met weekly, on Fridays or Wednesdays, in order to ensure that every member of our team was fully aware of the progress of the project as well as current goals. We established a project timeline, and whenever we met, we tried our best to check carefully in order to discuss what we had done individually, and to plan what each member should tackle for the following week. Often times making everyone appear at the meetings was a challenge, so our Project Manager compelled everyone to download and install Groupme (an immensely efficient mobile group messaging app) to establish healthy communication flow and to make sure everyone is up-to-date with the project. We used (and eventually abused) GroupMe until we became friends!

4.1.2 Specification

Once we created our first version of LRM, it became a standalone specification for the BuckCal language. Each time we got stuck in something underspecified or incorrectly specified, we referred to the sacred piece of document--the LRM. To make sure we had constant faith on our LRM, we kept the document up-to-date. We grew along with our LRM and eventually BuckCal was born.

4.1.3 Development

Although it might seem to be in disarray, but actually we effectively maintained a parallel processes for developing each feature of BuckCal. First, we attempted to create an elementary version of the desired feature (MVP - Minimum Viable Product) as soon as possible. Upon completion, while we tested the feature either by printing out results, writing small test code, we were able to identify few areas of improvement for each feature along the way. As we continued to develop and test our feature, it was easy to figure out areas of improvement and decide on which of these was feasible and whether

implementing additional feature was plausible. When these little improvements developed into a full-fledged feature, we were eventually able to run full integration tests, making the features bug-free.

4.1.4 Testing

At a high level, we made sure to allow feedback at all steps in our design process. To do this, we designed and redesigned our testing frameworks, modularized testing processes, and tested as soon as any feature was implemented in the language. This helped us maintain testing capabilities for each feature before our full integration tests were ready for the full-fledged language. We'll discuss our testing process at great lengths in section 6.

4.2 Style Guide

We used the following rules when writing our code to ensure maximum readability:

- Use recursion as much as possible to take advantage of the wonderful functional programming language called OCaml
- Each line of code should remain under 100-110 characters
- Use comments as much as possible for each critical section of code
- Use meaningful variables and function names
- Use a fancy Text Editor (i.e. Sublime Text 2) to make our code colorful and therefore easily readable and quickly writable (often taking advantage of its auto-complete functionality).

4.3 Team Responsibilities

In order to facilitate splitting up work, we first identified our interfaces between sections. We then made sure to get everyone onboard with git hosted by GitHub. Version control was essential to making sure everyone could work without stepping on each other's toes. With these set in stone, parallelizing was not a problem. We split up our group into multiple smaller more focused teams (although our work often overlapped, since we worked together).

- Front End: Meng Wang, Ahmad Maruf, Prachi Shukla (Scanner, parser, type inference, Code Generation, User-level library)
- Back End: Meng Wang, Lingyuan He (C++ implementation, Built-in library, distributed computing)
- Testing: Lan Yang
- Report/Writing/Editing/Compiling: Ahmad Maruf
- Debugging: Everyone on a rolling basis

4.4 Project Timeline

Our timeline was carefully laid out from the start:
Sep 24th Proposal due date
Oct 7th BuckCal syntax created
Oct 14th Scanner/parser unambiguous and working
Oct 20th LRM first draft
Oct 27th LRM due date
Nov 10th Early version of Architectural design
Nov 22nd Architectural design finalized
Dec 14th Compiler works, all tests pass
Dec 15th Final project report
Dec 16th Final project Slides
Dec 17th Final project due date

4.5 Development Environment

The BuckCal team developed on a variety of environments including mac OS X, Ubuntu, and Windows 7. We used OCaml version 4.00.1, OCamllex, and OCamlyacc for the compiler itself. We used git hosted on GitHub for version control. Lastly, we used bash scripts and makefiles to ease the work of compiling and testing the code.

4.6 Project Log

Please see Appendix (section 8.3.1.) for our project log from GitHub (authored and dated).

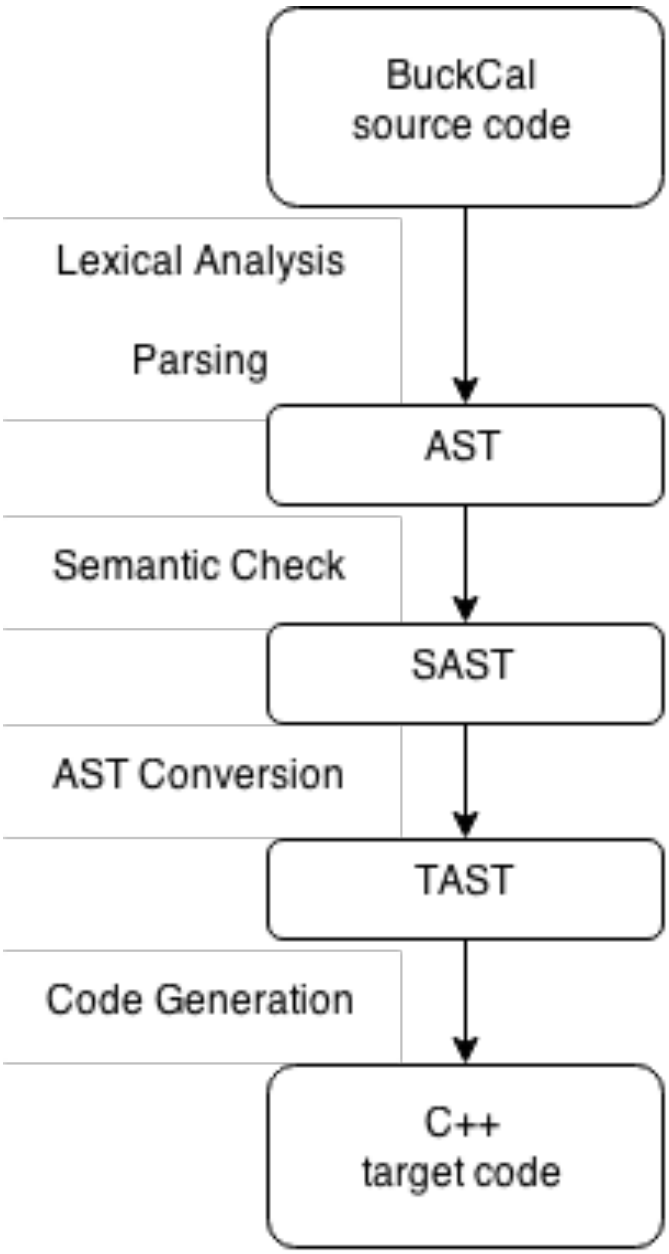
Chapter 5

Architectural Design

"If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization."
– Gerald Weinberg (American computer scientist)

5.1 Architecture of BuckCal translator

The architectural design of BuckCal translator is illustrated in the following diagram:



No doubt, we worked as a team. Everyone gave a hand with each other's tasks at all different levels of our project's progress. Yet as a part of the project requirement, we assigned particular roles. Just to give an idea, scanning, parsing, semantic checking, and lexical analyzing were all handled mostly Meng Wang and some by Lingyuan He; C++ code generation and implementing BuckCal user-level library were handled by Ahmad Maruf; rest of the built-in library and all other C++ implementation were handled by Lingyuan He; and some implementation and most of the debugging were done by Lan Yang and Prachi Shukla.

5.2 Lexical Analysis

The BuckCal scanner tokenizes the input. Whitespaces (including space, newline and tab) and comments are discarded. Illegal character combinations, such as malformed escape sequences, are caught and a Scanner_error exception is raised. The scanner is in src/scanner.mll.

5.3 Parsing and AST

The parser generates an abstract syntax tree (AST) from the tokens provided by the scanner. Syntax errors are caught here. The parser is in src/parser.mly, and AST data types are defined in src/ast.ml. An AST of a BuckCal program contains function definitions, top-level variable declarations and top-level statements. Meanwhile, all imports are resolved and the function definitions in imported source files are combined into local function definition list. Now, the AST is ready for semantic check.

5.4 Semantic check and SAST

In semantic check, the AST is walked from top to bottom. First, the function definitions are checked to assemble a function table. BuckCal support function overloading, so different functions should have different function signatures. User defined functions cannot not be identical with built-in functions. Second, the top-level variable declarations are checked to assemble a top-level variable table. Then, the top-level statements are checked: type of each expression is inferred, each variable identifier is checked against variable table and each function call is checked against function table. Type mismatching, undefined variable and unimplemented functions are detected. Finally, a type-safe, semantically checked AST (SAST) is generated. The semantic check functions are defined in src/scheck.ml, and the TAST data types are defined in src/sast.ml.

5.5 AST Conversions and TAST

The SAST is converted into target language AST (TAST) to make code generation easier. Some grammar structures in BuckCal have higher abstraction level than target language C++, such as matrix literal, matrix subscripting and counting loop: temporary C++ variables and corresponding declarations are generated to help with keeping the conversion context-free, and runtime checking statements are generated to validate

arguments. All temporary variables generated in this step is named with capitalized letter to prevent conflicts with user defined variables, and a counter is maintained all around to prevent naming conflicts between temporary variables. The block naming scope in C++ is also utilized. Also, all user-defined functions named “main” are renamed to “B_main” to avoid naming conflicts. Finally, the top-level variables and statements are combined into C++ main() function definition. The conversion function is defined in translate.ml, and a simplified C++ AST is defined in tast.ml.

5.6 Code Generation

With a TAST, C++ code can be generated. Each statement in TAST in translated into one line of C++ code, and an helper C++ library header file is included. The code generator is in src/codegen.ml, and all C++ library functions are put in lib/ directory.

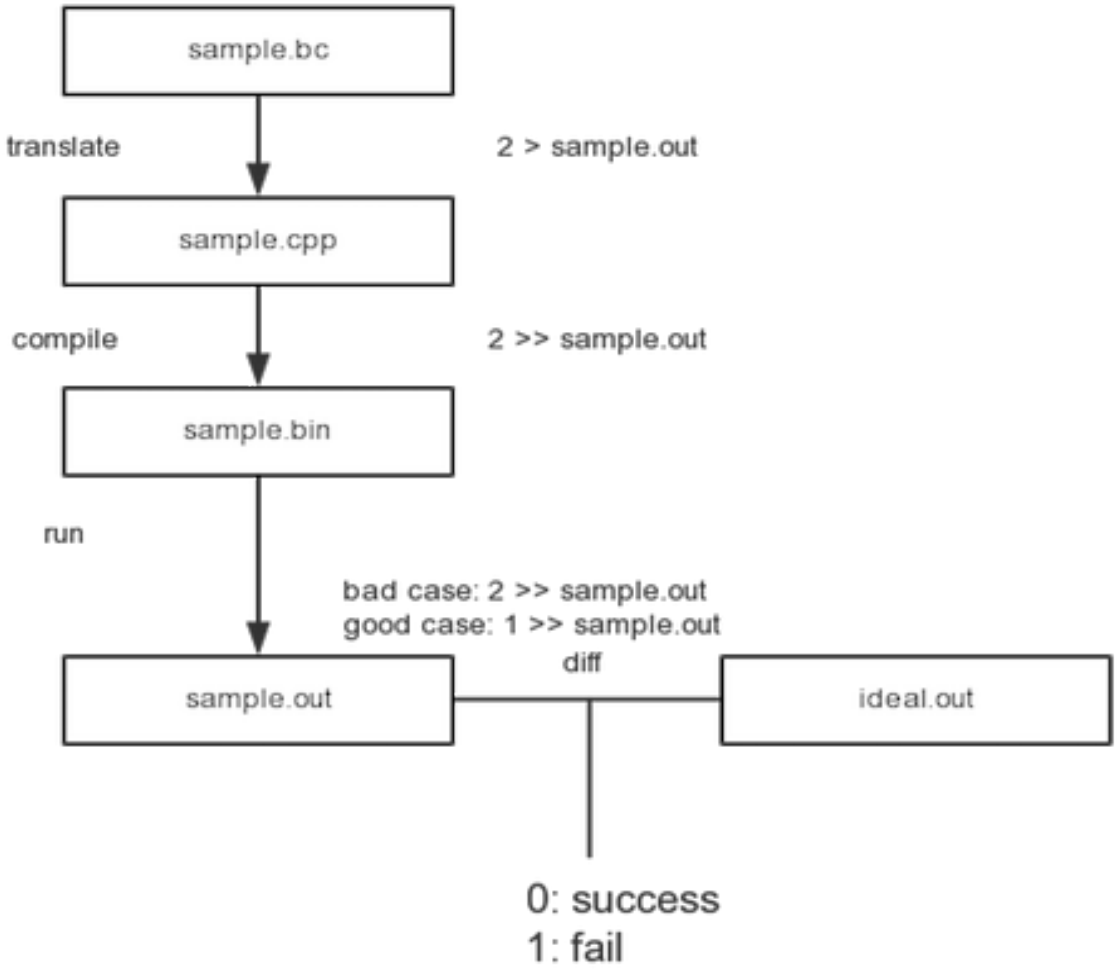
Chapter 6

Test Plan

"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. Kernighan (Canadian Computer Scientist, Co-author of "C programming language")

6.1 Test Structure

Our testing framework includes two parts: good case test & bad case test.



Individual tests are:

- Keywords
- Identifier
- Operators and separators
- Auto data type conversion between **int** and **double**
- Variable assignment and comparison
- String operations
- Arithmetic operations
- **if/else** statements
- **if/elif/else** statements
- Counting loops
- Conditional loops
- Function declaration
- Function calling
- Function overloading
- Function call
- **import**
- Matrix definition
- Matrix accessing
- Matrix subscription
- Built-in library: rows & cols
- Built-in library: string_of_int, string_of_double, int_of_string, double_of_string
- Built-in library: mat_int_of_string, mat_double_of_string, mat_string_of_int, mat_string_of_double
- Built-in library: rowname & colname
- Built-in library: colcat & rowcat
- Built-in library: strlen & slice
- Built-in library: getrow & getcol & setrow & setcol
- Built-in library: init_mat
- User-level library: range & range_col
- User-level library: abs
- User-level library: sum_row & sum_col
- User-level library: avg_row & avg_col

6.2 Example Tests

Example 1
Bad case: sample43.bc

```
#Test matrix definition
int mat ax: {1, 2, 3; 4, 5, 6; 7, 8, 9};
int a;
a: ax[2, 4];
disp a;
```

Output of badsample43: badsample43.cpp

```
#include "buckcal_mat.hpp"
using namespace std;
int main() {
try {
int T_0_0[] = { 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 };
int_mat TT_0_0 = ( int_mat( T_0_0 , 3 , 3 ) );
int_mat ax =  TT_0_0 ;
int a =  0 ;
if (!(( ( ( ( 2 - 1 ) >= 0 ) && ( ( 2 - 1 ) < ( rows( ax ) ) ) ) && ( ( ( 4 - 1 ) >= 0 ) && ( ( 4 - 1 ) < ( cols( ax ) ) ) ) ))) throw invalid_argument("Matsub index check");
( a = (ax[( ( ( 2 - 1 ) * ( cols( ax ) ) ) + ( 4 - 1 ) )]) ) ;
cout << a << endl;
return 0 ;
} catch (exception & e) { cerr << e.what() << endl; }
}
```

Execution:

```
matsub: index check failed, bad index
```

Example 2
Good case: sample26.bc

```
int mat ax;
double mat dx;
string mat sx: {'1.1','1.2';'1.3','1.4';'1.5','1.6'};
ax: mat_int_of_string(sx);
```

```
disp ax;
dx: mat_double_of_string(sx);
disp dx;
ax: {1,2,3;4,5,6;7,8,9};
dx: {1.1,2.2;3.3,4.4};
sx: mat_string_of_int(ax);
disp sx;
sx: mat_string_of_double(dx);
disp sx;
dx: mat_double_of_int(ax);
disp dx;
dx: {1.1,2.2;3.3,4.4};
ax: mat_int_of_double(dx);
disp ax;
```

Output of goodsample26: goodsample26.cpp

```
#include "buckcal_mat.hpp"
using namespace std;
int main() {
try {
int T_0_0[] = {};
int_mat TT_0_0 = ( int_mat( T_0_0 , 0 , 0 ) );
int_mat ax = TT_0_0 ;
double T_1_0[] = {};
double_mat TT_1_0 = ( double_mat( T_1_0 , 0 , 0 ) );
double_mat dx = TT_1_0 ;
string T_2_0[] = { string("1.1") , string("1.2") , string("1.3") , string("1.4") , string("1.5") ,
string("1.6") };
string_mat TT_2_0 = ( string_mat( T_2_0 , 3 , 2 ) );
string_mat sx = TT_2_0 ;
( ax = ( mat_int_of_string( sx ) ) ) ;
cout << ax << endl;
( dx = ( mat_double_of_string( sx ) ) ) ;
cout << dx << endl;
int T_12_0[] = { 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 };
int_mat TT_12_0 = ( int_mat( T_12_0 , 3 , 3 ) );
( ax = TT_12_0 ) ;
double T_13_0[] = {1.100000 , 2.200000 , 3.300000 , 4.400000};
double_mat TT_13_0 = ( double_mat( T_13_0 , 2 , 2 ) );
( dx = TT_13_0 ) ;
( sx = ( mat_string_of_int( ax ) ) ) ;
```

```
cout <<  sx  << endl;
(  sx  = ( mat_string_of_double( dx ) ) ) ;
cout <<  sx  << endl;
(  dx  = ( mat_double_of_int( ax ) ) ) ;
cout <<  dx  << endl;
double T_23_0[] = {1.100000 , 2.200000 , 3.300000 , 4.400000};
double_mat TT_23_0 = ( double_mat( T_23_0  ,  2  ,  2 ) );
(  dx  =  TT_23_0  ) ;
(  ax  = ( mat_int_of_double( dx ) ) ) ;
cout <<  ax  << endl;
return  0  ;
} catch (exception & e) { cerr << e.what() << endl; }
}
```

Execution: (Output)

```
      c1      c2
r1    [1,     1]
r2    [1,     1]
r3    [1,     1]

      c1      c2
r1    [1.1,   1.2]
r2    [1.3,   1.4]
r3    [1.5,   1.6]

      c1      c2      c3
r1    [1,     2,      3]
r2    [4,     5,      6]
r3    [7,     8,      9]

      c1      c2
r1    [1.1,   2.2]
r2    [3.3,   4.4]

      c1      c2      c3
r1    [1,     2,      3]
r2    [4,     5,      6]
r3    [7,     8,      9]

      c1      c2
r1    [1,     2]
r2    [3,     4]
```

Chapter 7

Lessons Learned

"Just because something ends doesn't mean it never should've been. Remember, you lived, you learned, you grew, and you moved on."
~ Unknown

7.1 Ahmad Maruf

First of all, this class has been an amazing learning experience. It was my first class project at Columbia University in which I felt so engaged, thrilled (and at times, stressed!), and overall, a fulfilling sense of ownership. As the Project Manager, I learned many important lessons. For example——

- Know your deadline, prioritize, and know what features to exclude:
  Sometimes it's better to say "no" to an additional feature than falling behind the deadline in implementing it. Because of the time constraint, we had to discard many fancy features we initially thought we'd implement. But there's no room for regret, because after all, I believe this class is more about understanding the core structures of a language & translator than trying to impress people with fancy features.
- Delivering good work is not a dictatorship:
  Distributing different parts of the project to different teammates is of course important, but evenly distributing the work is more important. Evenly distributing the ownership of different modules of the project would not only allow works to be done in parallel, but also make debugging easier. Although a successful project requires each teammate having a comprehensive knowledge about the project as a whole, an uneven distribution of workload results in some teammates knowing different parts of the project way more than others just because of their involvement in them. In our case, partly due to a tight timeline, only a few people were building the compiler. The rest of us spent much time in trying to understand the code written by others when debugging, resulting in slow progress.
- Find the right people to do the right thing:
  It's also true that it was a challenge for me to evenly distribute the workload of our project, because everyone has got different skill sets. I often had to juggle between writing the reports, writing snippets of codes to implement and test different parts, and making sure everyone is communicating properly (with each other as well as with the TA). It was not easy, but it was fun. Furthermore, not everyone is good at implementation, not only because of the unusual ways of functional programming in OCaml (that a lot of us were not used to), but also for

not having the same analytical skills or perspective. Likewise, testing is also not for everyone; it requires patience and comprehensive planning. So, it would've been more efficient to honestly evaluate people's skill sets before the team members seriously got their hands dirty.

7.2 Lan Yang

As a Tester in the team, there are three things that I learned most:

- A robust, complete, and automatic testing framework should be built as early as possible. I had to modify the testing framework twice, which wasted a lot of time. First time, I modified it to separate procedures for good cases and bad cases. Second time, I added runtime error to case outputs. Test script is not hard to write when you figure out the whole testing process. The key point is to have a good test procedure.
- Keep track of every implementation and improvement of project codes and write corresponding tricky but as detailed as possible test cases. For example, to test function definitions, I wrote test cases to test the return values, function arguments, and function implementation contents separately.
- Classify all test cases: Aside from separating good cases with bad cases, I classified every case according to its test intention. It helps make testing much more efficient and fathomable for other teammates.

7.3 Lingyuan He

Throughout the project, our learning progress has been non-stop. Team working in an integrated compiler project, as expected, turns out to be both fascinating and challenging. After this project, although I am not the manager, I still have learnt many new things in working with people.

- It is more about making a timeline:
  In the end of the day, as anticipated, I am going to say, "start early". But how, exactly? In this project, I feel that making a timeline should have helped more. As everybody has been busy all semester, we haven't done much until late in the last weeks. We did agree to make our scanner and parser around LRM due time, and we completed it. However, this practice was not enforced, but it should have been in place even earlier.
- Get people to work in small teams:
  It was no fun getting all five people meeting, obviously. So I suggested to pair people into implementation and testing teams with some rotation. This ease the time consuming part of coordinating all people together, and also facilitate good partial testing practice. It turns out to be a good method, but was put into place a little bit late. I believe it will be of more use earlier in regular weeks.
- Contribute to the team, but also work as a team:
  What I have learnt more about working in a group is that, in addition to do your work, you have to function as a member rather than just an individual. That is, to keep communicating with people while working effectively. You need to work

with people to get your idea across, listen to others and find the best solution together. And you also need to talk to the person that knows best in a part rather than go through everything yourself, especially when debugging. Meanwhile, spending time and making real progress are still essential, that makes sure you are contributing.

Overall, it has been a great experience working with a team on a compiler project; I believe it has taught me more on doing Project with people than an implementation itself. But of course, the actual coding part is fun and practical.

7.4 Meng Wang

As for the programming part, I think it's a good style to keep the modules small. Especially, don't make one big module, which implemented several different functionalities. When I found that the code generator was both handling the structure and syntax of target language, I decided that a target AST is necessary, and the AST conversion should be split from code generation. In this way, the difficulty in implementing both modules becomes lower and the architecture of BuckCal translator is clearer.

As a language guru, it's important that you always keep the feature of your language in mind. When your team is try to make progress, don't easily give up your feature only because of compliance like "Oh, it looks difficult to implement" - most of the time, in fact, it isn't.

And choosing a proper target language is also important. Thanks to Prof. Edwards' for his suggestions, we used C++ as the target language, which brought to me a lot of challenges and fun in designing the translation framework. If we had used R (which was our very first idea) instead of C++, I would've greatly regretted it; because R was so similar to BuckCal that we wouldn't have had the chance to experience all the wonderful.

7.5 Prachi Shukla

The PLT class as well as the project has been one learning experience for me. It's really exciting to see how the compiler translates one higher-level language down to the next lower level until the code is generated in assembly-level language. It would probably sound like stating an obvious fact, but I feel it's very intricate and I'll shortly get to why I'm saying that. Our team had first decided to proceed with 'R' as the code generation target language, which looked very similar to the proposed 'BuckCal' language, and I too initially seemed to be fine with it. But it later struck me that the compiler is supposed to translate from high level to low level; and if we proceeded with 'R' we would be rendering this concept void. I then put it across my team and then we finally went and spoke with the Professor. And luckily, we got the right target language – C++, which is indeed a lower level language than BuckCal with no syntax resemblance. This is when I really put to use the intricate functionality of a compiler. I also feel it's extremely important to work with the right people. I'm very lucky to have gotten hard-working teammates and without their support this project would not be a success. We all communicated well and kept the project going. Also, we kept seeing Kuangya (TA we

were assigned to) weekly, and that way we ended up at least partially meeting the weekly deadlines that we would set for our project.

Chapter 8

Appendix

"An appendix is something found in the back of a book.
Sometimes they get in people and have to be taken out."
~ Unknown

8.1. Appendix A

8.1.1. BuckCal Library

8.1.1.1. Built-in Library Functions: Implemented in C++

Functions (Grouped by Category)	Description

string string_of_int(int x);
string string_of_double(double x);
int int_of_string(string x);
double double_of_string(string x);
	Convert primitive data types.
	
	int and double can be converted implicitly.

int_mat mat_int_of_string(string_mat x);
double_mat
mat_double_of_string(string_mat x);
string_mat mat_string_of_int(int_mat x);
string_mat
mat_string_of_double(double_mat x);
int_mat mat_int_of_double(double_mat x);
double_mat mat_double_of_int(int_mat x);
	Convert different types of matrices.

int rows(int_mat mx);
int rows(double_mat mx);
int rows(string_mat mx);
	Get number of rows.

int cols(int_mat mx);
int cols(double_mat mx);
int cols(string_mat mx);
	Get number of columns.

int_mat rowcat(int_mat mx1, int_mat mx2); double_mat rowcat(double_mat mx1,	Concatenate two matrices by rows, column number must match for non-empty matrices.

double_mat mx2);
string_mat rowcat(string_mat mx1, string_mat mx2);
int_mat rowcat(int_mat mx1, double_mat mx2);
double_mat rowcat(double_mat mx1, int_mat mx2);
	int mat and double mat can be mixed for programmer's convenience, but the first argument will decide output matrix type.

int_mat colcat(int_mat mx1, int_mat mx2);
double_mat colcat(double_mat mx1, double_mat mx2);
string_mat colcat(string_mat mx1, string_mat mx2);
int_mat colcat(int_mat mx1, double_mat mx2);
double_mat colcat(double_mat mx1, int_mat mx2);
	Concatenate two matrices by columns, row number must match for non-empty matrices.
	
	int mat and double mat can be mixed for programmer's convenience, but the first argument will decide output matrix type.

void rowname(int_mat &mx, string_mat n);
void rowname(double_mat &mx, string_mat n);
void rowname(string_mat &mx, string_mat n);
	Set row names according to n, number of entries in n must match mx's row number.

void colname(int_mat &mx, string_mat n);
void colname(double_mat &mx, string_mat n);
void colname(string_mat &mx, string_mat n);
	Set column names according to n, number of entries in n must match mx's column number.

int strlen(string x); string slice(string x, int l, int r);	String operations, strlen returns number of characters slice returns a substring (start character number l to (r-1))

int_mat getrow(int_mat mat, int r);
double_mat getrow(double_mat mat, int r);
string_mat getrow(string_mat mat, int r);
	Get row r from the matrix, column and row names will be returned as well.

void setrow(int_mat mat, int r, int_mat set); void setrow(double_mat mat, int r, double_mat set);	Set row r of the matrix, column number must match, row name will be transferred, but not column names. int mat and double mat can be mixed for

void setrow(double_mat mat, int r, int_mat set);
void setrow(int_mat mat, int r, double_mat set);
void setrow(string_mat mat, int r, string_mat set);
	programmer's convenience, the matrix to be changed will retain its type.

int_mat getcol(int_mat mat, int c);
double_mat getcol(double_mat mat, int c);
string_mat getcol(string_mat mat, int c);
	Get column c from the matrix, column and row names will be returned as well.

void setcol(int_mat mat, int c, int_mat set);
void setcol(double_mat mat, int c, double_mat set);
void setcol(double_mat mat, int c, int_mat set);
void setcol(int_mat mat, int c, double_mat set);
void setcol(string_mat mat, int c, string_mat set);
	Set column c of the matrix, row number must match, column name will be transferred, but not row names.
	
	int mat and double mat can be mixed for programmer's convenience, the matrix to be changed will retain its type.

int_mat init_mat(int r, int c, int init);
double_mat init_mat(int r, int c, double init);
string_mat init_mat(int r, int c, string init);
	Return an initialized r row c column matrix with the initial values provided. Column and row names are default.

8.1.1.2. User-Level Library Functions: Implemented in BuckCal

Function (Grouped by Category)	Description
int mat range(int x, int y);	Return a row vector {x, x+1, … , y}. If y < x, return {}.
int mat range_col(int x, int y);	Return a row vector {x, x+1, … , y}. If y < x, return {}.
double abs(double x);	Returns absolute values, int can also use this function due to implicit conversion.
double mat sum_row(double mat mx);	Returns a row vector that has the sum of all the rows on each column of the matrix, int matrix can also use this function due to implicit conversion.
double mat sum_col(double mat mx);	Returns a column vector that has the sum of all the columns on each row of the

	matrix, int matrix can also use this function due to implicit conversion.
double mat avg_row(double mat mx);	Returns a row vector that has the average of all the rows on each column of the matrix, int matrix can also use this function due to implicit conversion.
double mat avg_col(double mat mx);	Returns a column vector that has the average of all the columns on each row of the matrix, int matrix can also use this function due to implicit conversion.

8.2 Appendix B

8.2.1. Code Listing

This appendix contains the code listing for BuckCal. Line counts are not included for test scripts, Makefiles, and BuckCal test cases.

"Measuring programming progress by lines of code is like measuring aircraft building progress by weight."
~ Bill Gates (Co-founder of Microsoft)

8.2.1.1. Summary: OCaml Code

File Name	Description	Lines of Code
src/scanner.mll	Token scanner	102
src/parser.mly	Lexical parser, generates AST	195
src/ast.ml	Initial AST definition	93
src/scheck.ml	Semantic check, generates SAST	303
src/scheck_expr.ml	Helpers for semantic check	206
src/sast.ml	Safe AST definition	64
src/translate.ml	Generate target AST	176

src/translate_expr.ml	Helpers for translate	48
src/tast.ml	Target AST definition, for generating C++	65
src/codegen.ml	Generates C++ code	111
src/lib.ml	Built-in library call declarations	189
src/main.ml	Main compiler executable	88

8.2.1.2.Summary: C++ Code

"C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do, it blows away your whole leg."
~ Bjarne Stroustrup
(Danish Computer Scientist, Developer of the C++ programming language, and a Visiting Professor at Columbia University)

File Name	Description	Lines of Code
lib/buckcal_mat.hpp	Underlying matrix data types, and built-in function declarations	175
lib/buckcal_mat.cpp	Implementation of matrix data types and related functions	390
lib/buckcal_core.cpp	Implementation of built-in functions	526

8.2.1.3. Summary: BuckCal Code

File Name	Description	Lines of Code
lib/buckcal_lib.bc	User-level library functions	101

8.3. Appendix C

8.3.1. Project Log

12/15/14	Ahmad Maruf	OMG Last Commit! Yay!
12/15/14	mwcu	merge
12/15/14	Ahmad Maruf	usr folder
12/15/14	mwcu	remove sample.bin in make clean
12/15/14	Ahmad Maruf	added sample.bc and imp.bc for project demo
12/15/14	mwcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/15/14	mwcu	import: bfs->dfs
12/15/14	Lingyuan He	fix bad cases ideal output
12/15/14	mwcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/15/14	mwcu	supress warning 4
12/15/14	Lingyuan He	fix range row/col name
12/14/14	Meng Wang	Add README in lib/
12/14/14	Meng Wang	README
12/14/14	Lingyuan He	fix sample 54
12/14/14	Lingyuan He	usr folder
12/14/14	Lingyuan He	fix test.sh
12/14/14	Lingyuan He	fix README
12/14/14	Lingyuan He	minor fix
12/14/14	Lingyuan He	restructure folders
12/13/14	mwcu	fix test.sh
12/13/14	mwcu	fix test.sh
12/13/14	mwcu	fix test.sh
12/13/14	mwcu	fix test.sh
12/13/14	Lan Yang	Merge branch 'master' of https://github.com/Maruf789/PLT
12/13/14	Lan Yang	test
12/13/14	mwcu	mkdir in test
12/13/14	Lan Yang	Merge branch 'master' of https://github.com/Maruf789/PLT
12/13/14	Lan Yang	test
12/13/14	mwcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/13/14	mwcu	try..catch in main
12/13/14	Lingyuan He	Merge branch 'master' of https://github.com/Maruf789/PLT
12/13/14	Lingyuan He	cleanup c++ and lib.ml void fix
12/13/14	Lan Yang	test
12/13/14	Lan Yang	test
12/13/14	Prachi Shukla	updated goodsample ideal output files
12/13/14	Prachi Shukla	test_samples_good
12/13/14	mwcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/13/14	mwcu	init intmat with doubke mat
12/13/14	Lingyuan He	Merge branch 'master' of https://github.com/Maruf789/PLT
12/13/14	Lingyuan He	lib.ml and c++ cleanup

12/13/14	Lan Yang	Merge branch 'master' of https://github.com/Maruf789/PLT
12/13/14	Lan Yang	test
12/13/14	Lingyuan He	Merge branch 'master' of https://github.com/Maruf789/PLT
12/13/14	Lingyuan He	update slice
12/13/14	mwcu	fix built-in function empty
12/13/14	mwcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/13/14	mwcu	fix uniop Neg
12/13/14	Lan Yang	Merge branch 'master' of https://github.com/Maruf789/PLT
12/13/14	Lan Yang	test
12/13/14	mwcu	error if call a function not implemented
12/13/14	mwcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/13/14	mwcu	avoid name conflict of 'main'
12/13/14	Lingyuan He	row col get names
12/13/14	Lingyuan He	set/get col and fix header
12/13/14	Lan Yang	test
12/13/14	Lan Yang	test
12/13/14	Lan Yang	test
12/13/14	Lan Yang	Merge branch 'master' of https://github.com/Maruf789/PLT
12/13/14	Lan Yang	modify test arch
12/13/14	Lingyuan He	cleanup
12/13/14	Lingyuan He	get/setrow and init_mat
12/13/14	Lingyuan He	get/setrow and init_mat
12/13/14	Lan Yang	test
12/13/14	Lingyuan He	change sample 12
12/13/14	Lingyuan He	c++ print tweak and sample 20 changed
12/13/14	Lingyuan He	fix typo in colcat
12/13/14	mwcu	exit code fixed. 1 on failure
12/13/14	mwcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/13/14	mwcu	clean .cmx .o
12/13/14	Ahmad Maruf	Updated test.sh
12/13/14	Ahmad Maruf	added sample tests
12/13/14	Ahmad Maruf	updated buckcal_lib.bc
12/13/14	Ahmad Maruf	added sample for buckcal_lib.bc functions
12/12/14	Lingyuan He	fix row/colcat, range, range_col, add sample56
12/12/14	Lingyuan He	fix row/col cat
12/12/14	Ahmad Maruf	sample38 updated
12/12/14	Ahmad Maruf	Merge branch 'master' of https://github.com/Maruf789/PLT
12/12/14	Ahmad Maruf	sample8 for good_test
12/12/14	mwcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/12/14	mwcu	minor fix
12/12/14	Lingyuan He	matsub check in translate and fix bc lib
12/12/14	mwcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/12/14	mwcu	fixing submat type check

12/12/14	Lingyuan He	fix dev_test
12/12/14	Lingyuan He	fix dev_test
12/12/14	Lingyuan He	fix dev_test
12/12/14	mwcu	update README
12/12/14	mwcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/12/14	Meng Wang	Delete buckcal_mat.hpp
12/12/14	Meng Wang	Delete buckcal_lib.bc
12/12/14	mwcu	aa
12/12/14	mwcu	use absolute path in test
12/12/14	Ahmad Maruf	Merge branch 'master' of https://github.com/Maruf789/PLT
12/12/14	Ahmad Maruf	updated buckcal_lib.bc
12/12/14	Lan Yang	Merge branch 'master' of https://github.com/Maruf789/PLT
12/12/14	Lan Yang	test
12/12/14	mwcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/12/14	mwcu	fix issue 55
12/12/14	Lingyuan He	dev_test makefile fix
12/12/14	mwcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/12/14	mwcu	fix issue 55
12/12/14	Lan Yang	Merge branch 'master' of https://github.com/Maruf789/PLT
12/12/14	Lan Yang	test
12/12/14	Lingyuan He	fix bad operand +
12/12/14	Lingyuan He	fix bad operand +
12/12/14	Lingyuan He	lib.ml
12/12/14	Lingyuan He	more cleanup
12/12/14	Lingyuan He	cleanup a bit
12/12/14	Lan Yang	Merge branch 'master' of https://github.com/Maruf789/PLT
12/12/14	Lan Yang	test
12/12/14	Lingyuan He	cleanup a bit
12/12/14	Lingyuan He	dev_test and renaming
12/12/14	Lingyuan He	dev_test and renaming
12/12/14	mwcu	clean dev_test
12/12/14	mwcu	add runtime check to matsub
12/12/14	mwcu	fix string mat add
12/12/14	mwcu	fix issue 50
12/12/14	mwcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/12/14	mwcu	show filename in error message
12/12/14	mwcu	shoe filename in error message
12/12/14	mwcu	dev_test
12/12/14	mwcu	add dev_test
12/12/14	mwcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/12/14	mwcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/12/14	mwcu	Merge branch 'import_dev' into mat_dev

12/12/14	mwcu	import done
12/12/14	Lan Yang	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/12/14	Lan Yang	test
12/12/14	Prachi Shukla	added functions avg_col, avg_row
12/12/14	Ahmad Maruf	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/12/14	Ahmad Maruf	updated
12/12/14	Prachi Shukla	added functions avg_col, avg_row
12/12/14	Lan Yang	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/12/14	Lan Yang	test
12/11/14	Lingyuan He	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	Lingyuan He	issue #52
12/12/14	Ahmad Maruf	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/12/14	Ahmad Maruf	implemented more functions
12/12/14	Lan Yang	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/12/14	Lan Yang	test
12/12/14	mwcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/12/14	mwcu	fix ftbl
12/12/14	mwcu	tmp commit
12/11/14	Ahmad Maruf	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	Ahmad Maruf	implemented some functions
12/11/14	Lan Yang	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	Lan Yang	test.sh
12/11/14	Lan Yang	merge
12/11/14	Lan Yang	merge
12/11/14	Lingyuan He	fix conflict
12/11/14	Lingyuan He	row/colname, string op
12/11/14	Lan Yang	lib
12/11/14	mwcu	fix init a int with submat
12/11/14	Lan Yang	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	Lan Yang	test
12/11/14	Ahmad Maruf	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	mwcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT

		into mat_dev
12/11/14	mwcu	parser done
12/11/14	Lingyuan He	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	Lingyuan He	merge
12/11/14	Ahmad Maruf	ADDED TWO LIBRARY FILES
12/11/14	Lan Yang	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	Lan Yang	test
12/11/14	Lingyuan He	col/rowcat, slice, strlen
12/11/14	mwcu	Merge branch 'mat_dev' into import_dev
12/11/14	mwcu	remove some unsed variable
12/11/14	mwcu	isl fixed
12/11/14	Lan Yang	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	mwcu	don't check return in function declare
12/11/14	Lan Yang	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	Lan Yang	test
12/11/14	mwcu	fix std::invalid_argument
12/11/14	Lan Yang	test
12/11/14	mwcu	half way import
12/11/14	Lan Yang	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	Lingyuan He	fix libarary
12/11/14	Lan Yang	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	Lan Yang	test
12/11/14	Lingyuan He	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	Lingyuan He	fix libarary
12/11/14	mwcu	remove lib function declarations
12/11/14	Lan Yang	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	Lan Yang	test
12/11/14	Ahmad Maruf	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	Ahmad Maruf	Updated the scheck for function table
12/11/14	mwcu	merge
12/11/14	mwcu	fixing temp variablenaming conflict
12/11/14	Ahmad Maruf	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	Lingyuan He	new header with lib call declarations

12/11/14	Ahmad Maruf	added builtin and user defined func table separately
12/11/14	Lingyuan He	row and col name in cpp file
12/11/14	mwcu	fix cols
12/11/14	mwcu	small formatting in translate.ml
12/11/14	mwcu	Matsub: A[1, 2] done
12/11/14	Prachi Shukla	added check for missing return statement in function; not tested
12/11/14	Prachi Shukla	resolved the missing return statements in functions, haven't tested it yet
12/11/14	Lan Yang	test
12/11/14	mwcu	fix lib.cpp
12/11/14	Lan Yang	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	Lan Yang	lib
12/11/14	mwcu	fix tid in CntFor
12/11/14	mwcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	Lan Yang	lib
12/11/14	mwcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	mwcu	out channel to file now
12/11/14	Lan Yang	test
12/10/14	Lingyuan He	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/10/14	Lingyuan He	remove c++11 things
12/10/14	Ahmad Maruf	updated the CntFor Loop
12/10/14	Ahmad Maruf	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/10/14	Ahmad Maruf	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/10/14	Ahmad Maruf	implemented CntFor Loop
12/10/14	Lan Yang	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/10/14	Lan Yang	test
12/10/14	mwcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/10/14	mwcu	fix funcdef
12/10/14	Lan Yang	buckcal_lib.cpp
12/10/14	Lan Yang	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/10/14	Lan Yang	test
12/10/14	mwcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev

12/10/14	mwcu	add IIndex
12/10/14	Lingyuan He	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/10/14	Lingyuan He	fix test.sh
12/10/14	mwcu	
12/10/14	mwcu	merge
12/10/14	mwcu	fix ret type check
12/10/14	Prachi Shukla	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/10/14	Prachi Shukla	resolved return type mismatch
12/10/14	Lingyuan He	merge
12/10/14	Lingyuan He	library call and lib.ml modification
12/10/14	Lan Yang	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/10/14	Lan Yang	hhh
12/10/14	mwcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/10/14	mwcu	fix T naming conflict
12/10/14	Lan Yang	merge
12/10/14	Lan Yang	new test arch
12/10/14	Lingyuan He	lib functions as declaration only
12/10/14	Lingyuan He	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/10/14	Lingyuan He	test framework with matrix support
12/10/14	mwcu	fix .gitignore in /script
12/10/14	Lingyuan He	hhh
12/10/14	Ahmad Maruf	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/10/14	Ahmad Maruf	fixed the SIF function issue#31
12/10/14	Lingyuan He	fix array no []
12/10/14	Lan Yang	modify test.sh
12/10/14	Lingyuan He	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/10/14	Lan Yang	merge
12/10/14	Lan Yang	c++file improved
12/10/14	Lingyuan He	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/10/14	mwcu	fix int-double match in function call
12/10/14	Ahmad Maruf	Merge branch 'master' of https://github.com/Maruf789/PLT
12/10/14	mwcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/10/14	mwcu	trying submat
12/10/14	Lingyuan He	main function return

12/10/14	Lan Yang	test
12/10/14	Lan Yang	merge
12/10/14	mwcu	formatting
12/10/14	mwcu	remove Failure
12/10/14	mwcu	formating
12/10/14	mwcu	fix trans_expr
12/10/14	Ahmad Maruf	added the modified LRM
12/9/14	Lingyuan He	locals and test.sh
12/9/14	Ahmad Maruf	Deleted BuckCal_LRM.pdf
12/9/14	Prachi Shukla	applied the patch to scheck for break,continue and return
12/9/14	Ahmad Maruf	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/9/14	Ahmad Maruf	implemented SIf
12/9/14	Prachi Shukla	add checks for return type, break and continue in scheck.ml
12/9/14	Lan Yang	merge
12/9/14	mwcu	small bug
12/9/14	Lan Yang	test
12/9/14	mwcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/9/14	mwcu	fix (=)
12/9/14	Lan Yang	add two repositories
12/9/14	mwcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/9/14	mwcu	fix double quote in string
12/9/14	Lan Yang	merge
12/9/14	Lan Yang	sample2
12/9/14	mwcu	type equality: int = double
12/9/14	Lan Yang	merge
12/9/14	Lan Yang	merge
12/9/14	mwcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/9/14	mwcu	array code []
12/9/14	Lingyuan He	Disp
12/9/14	Lan Yang	merge
12/9/14	mwcu	CntFor vo.1
12/9/14	Lan Yang	merge
12/9/14	Lan Yang	merge
12/9/14	Lingyuan He	disp
12/9/14	mwcu	while end
12/9/14	mwcu	vo.2 done
12/9/14	Ahmad Maruf	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/9/14	Ahmad Maruf	implemented Array expression

12/9/14	Lan Yang	test
12/9/14	Lan Yang	merge
12/9/14	Lingyuan He	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/9/14	Lingyuan He	c++ matrix module
12/9/14	Lan Yang	merge
12/9/14	Lan Yang	merge
12/9/14	Lan Yang	test.sh
12/9/14	mwcu	done
12/9/14	Ahmad Maruf	updated statement portion
12/9/14	mwcu	gen_stmt v0.5
12/9/14	mwcu	gen_stmt
12/9/14	mwcu	merge
12/9/14	mwcu	done
12/9/14	Ahmad Maruf	Updated "translate statement list"
12/9/14	Lan Yang	ivar
12/9/14	Lan Yang	Merge branch 'mat_dev' of https://github.com/Maruf78/PLT into mat_dev
12/9/14	mwcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/9/14	mwcu	temp
12/9/14	Lan Yang	nothing
12/9/14	mwcu	rm ir.ml
12/9/14	mwcu	codegen v0.1
12/9/14	Ahmad Maruf	Deleted ir.ml
12/9/14	Lan Yang	test on translator
12/9/14	mwcu	translate v0.1
12/9/14	Lan Yang	modified test.sh
12/9/14	mwcu	nothing
12/9/14	mwcu	tmp
12/8/14	mwcu	merge
12/8/14	mwcu	fix issue #26
12/7/14	mwcu	test translate v0.1 - no mat support
12/7/14	mwcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/7/14	mwcu	first attempt on matval
12/7/14	Meng Wang	add column and row numbers to SMatval
12/5/14	mwcu	add C++ header. g++ compiles.
12/4/14	Meng Wang	translate v0.1 done
12/4/14	Meng Wang	trans_elif done
11/30/14	mwcu	translate stmts v0.1
11/30/14	Lan Yang	more test cases for sast
11/30/14	Lan Yang	sast test
11/30/14	Lan Yang	test.sh

11/30/14	mwcu	fix parser issues
11/30/14	mwcu	trans_expr v0.1
11/30/14	Lan Yang	modified test cases
11/30/14	mwcu	smart test
11/30/14	mwcu	make clean
11/30/14	mwcu	fix parser: function declare
11/30/14	mwcu	Merge branch 'test_scanner_parser' of https://github.com/Maruf789/PLT into test_scanner_parser
11/30/14	mwcu	scheck done
11/29/14	Lan Yang	most basic test cases done
11/29/14	Lan Yang	more test cases
11/29/14	mwcu	add library function
11/28/14	mwcu	function: take arguments into variable table
11/28/14	mwcu	scheck v0.5
11/26/14	Lingyuan He	svar_init_sexpr
11/23/14	Lan Yang	readme for test
11/23/14	Lan Yang	test cases v1
11/23/14	mwcu	fix tab
11/23/14	mwcu	Merge branch 'test_scanner_parser' of https://github.com/Maruf789/PLT into test_scanner_parser
11/23/14	Lan Yang	modified test.sh
11/23/14	mwcu	add \n to perror
11/23/14	mwcu	print error message to stderr now
11/23/14	mwcu	add comments in main
11/23/14	mwcu	Merge branch 'test_scanner_parser' of https://github.com/Maruf789/PLT into test_scanner_parser
11/23/14	mwcu	more precise error message
11/23/14	Lan Yang	test
11/23/14	mwcu	Add Scanner_error
11/23/14	mwcu	check_fundef v0.1
11/23/14	Lan Yang	merge
11/23/14	Lan Yang	retry test
11/23/14	Lan Yang	test
11/23/14	Lan Yang	first step for test cases
11/22/14	mwcu	check_sexpr v0.3
11/22/14	mwcu	scheck_expr v0.3
11/22/14	mwcu	scheck_expr v0.2
11/19/14	mwcu	ignore main.bin
11/19/14	mwcu	add helper function: find_var find_func
11/16/14	Lingyuan He	optimize Ast/Sast, change function coding style, use file input for main.bin, make will clean first, make test compile sampleo to R
11/15/14	mwcu	Add a README in src

11/14/14	mwcu	remove unused files
11/14/14	mwcu	can compile hello world
11/14/14	mwcu	split scheck.ml and translate.ml
11/13/14	mwcu	return type of expr
11/12/14	mwcu	translate v0.6
11/11/14	Ahmad Maruf	Renamed build_routine to build_routine.md
11/11/14	Ahmad Maruf	Updated README.md
11/11/14	Ahmad Maruf	Created this to move build_routine from README.md
11/10/14	mwcu	fix bug in string
11/10/14	mwcu	fix sampleo
11/10/14	mwcu	translate v0.5
11/10/14	mwcu	t
11/10/14	Lingyuan He	fix readme
11/10/14	Lingyuan He	clean folder structure
11/10/14	Lingyuan He	modify gitignore
11/10/14	Lingyuan He	fix sample error, call argument list and add empty statement
11/8/14	Meng Wang	main function v0.1
11/8/14	Meng Wang	Merge branch 'master' of https://github.com/Maruf789/PLT
11/8/14	Meng Wang	Error location
11/7/14	Meng Wang	Update README
11/7/14	Meng Wang	Update README
11/7/14	Meng Wang	Makefile and simple test
11/7/14	Meng Wang	format
11/7/14	Meng Wang	ddd
11/7/14	Meng Wang	parser done
11/7/14	Meng Wang	hmm
10/29/14	Meng Wang	merge
10/29/14	Meng Wang	parser v0.5 compiled now
10/27/14	Meng Wang	lexical error location done
10/27/14	Meng Wang	error location update
10/25/14	Meng Wang	Delete test_scanner.mll
10/25/14	Meng Wang	Delete scanner
10/25/14	Meng Wang	fix mysterious illegal characters
10/25/14	Meng Wang	remove WHILE
10/25/14	Meng Wang	test scanner v1.0
10/25/14	Lan Yang	fix problem in post_scanner
10/25/14	Lan Yang	README
10/25/14	Lan Yang	fix string problem in sample
10/25/14	Meng Wang	post
10/25/14	Lan Yang	string_lit buffer lexbuf
10/25/14	Lan Yang	two else, two return
10/25/14	Lan Yang	Merge branch 'master' of https://github.com/Maruf789/PLT
10/25/14	Lan Yang	problem with upper and lower case

10/25/14	Meng Wang	lower,upper
10/25/14	Lan Yang	fix buffer problem
10/25/14	Meng Wang	scanner test v0.5
10/25/14	Meng Wang	scanner
10/24/14	Ahmad Maruf	Rename LRM.md to LRM
10/24/14	Ahmad Maruf	Rename LRM to LRM.md
10/24/14	Ahmad Maruf	Created Language Reference Manual for BuckCal
10/24/14	Ahmad Maruf	Created LRM
9/20/14	Ahmad Maruf	Update Proposal.md
9/20/14	Ahmad Maruf	Update Proposal.md
9/20/14	Ahmad Maruf	Update Proposal.md
9/20/14	Ahmad Maruf	Rename Proposal to Proposal.md
9/20/14	Ahmad Maruf	Update Proposal
9/19/14	Ahmad Maruf	Update Proposal
9/19/14	Ahmad Maruf	Create Proposal
9/19/14	Ahmad Maruf	Create README.md