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 anyone 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 Appendix A, 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
```

Output:
The number is:
5

disp text; **disp** i; i : i + 1; **rof**	The number is: 6 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	Var (uppercase letter)
matrix01	number (number as the first letter)
food_day2	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; Scalar addition (matrix and value) and vector addition (matrix and matrix).
Minus '-'	Mathematical subtraction for numeric values. Scalar subtraction (matrix and value) and vector subtraction (matrix and matrix).
Multiply '*'	Mathematical multiply for numeric values. Scalar multiplication (matrix and value) and vector multiplication (matrix and matrix).
Divide '/'	Mathematical division for numeric values.

	Scalar multiplication (matrix and value) and vector division (matrix and matrix).
Assign ':'	Used in assignment and variable initialization.
Logical equal '=' and Logical unequal '!='	Used only for logical comparison, return a **bool.** Performed on primitive data types.
'<' '<=' '>' '>='	Tests if the left operand is less than/less than or equal to/ greater than/ greater than or equal to right operand. string is compared by dictionary order.
Index - '[ ]'	Used in matrix subscripting, subscript to one element only.
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.

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

Note that **int** and **double**, **int mat** and **double mat** can be converted implicitly, which means they can be used interchangeably. There are no **int←→double** conversion function, but **int mat←→double mat** conversion function does present.

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

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.

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 Appendix A). Users can also define their own.

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 (';').

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 not supported in BuckCal to keep it a simple language.

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, meaning that a function that is only declared cannot be called. As a result, recursion and mutual reference are not possible in BuckCal.

Arguments for the functions written in BuckCal are all passed by value. Some built-in functions do accept arguments by reference, for the ease of programming. 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 name of .bc file from current source file directory. Note that a semicolon is not needed after <filename>.

A normal multi-level import structure is supported in BuckCal. However, import instructions that result in a loop is not part of the specification of Buckcal, and it will cause erronous.

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. Program Structure

A complete BuckCal program consists of four parts in a fixed order.

1. *Import* instructions
   All the import instructions will appear first:
   import 'file1.bc'

import ‘file2.bc’

2. Function declarations or definitions
   All the function declarations and definitions to be used have to appear here.
3. Variable Declarations and definitions
   All variables to be used for the main script of the file will appear here. On-the-fly variable declarations are not supported.
4. Main script
   Main execution of the script, similar to a main() function. In an imported file, this part is ignored.

A BuckCal file can have all of the above parts, or just a selection of them in the given order.

3.13. Sample Programs

3.13.1. “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;
```

3.13.2. 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).

- Compiler Front End: scanner, parser
  Meng Wang, Lingyuan He
- Compiler Back End: semantic checking, translation and code generation
  Meng Wang, Ahmad Maruf, Prachi Shukla
- Library: C++ implementation, built-in library, user-level library
  Lingyuan He, Lan Yang
- Testing:
  Lan Yang

- Report/Editing:
  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 C for our project log from GitHub (authored and dated).

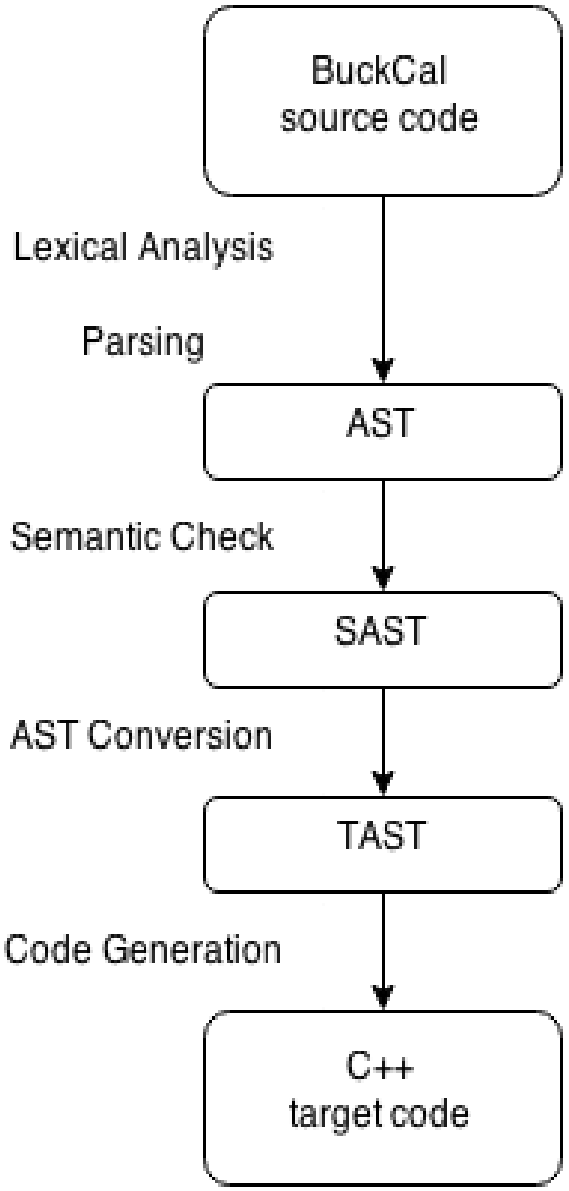
Chapter 5

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:



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.

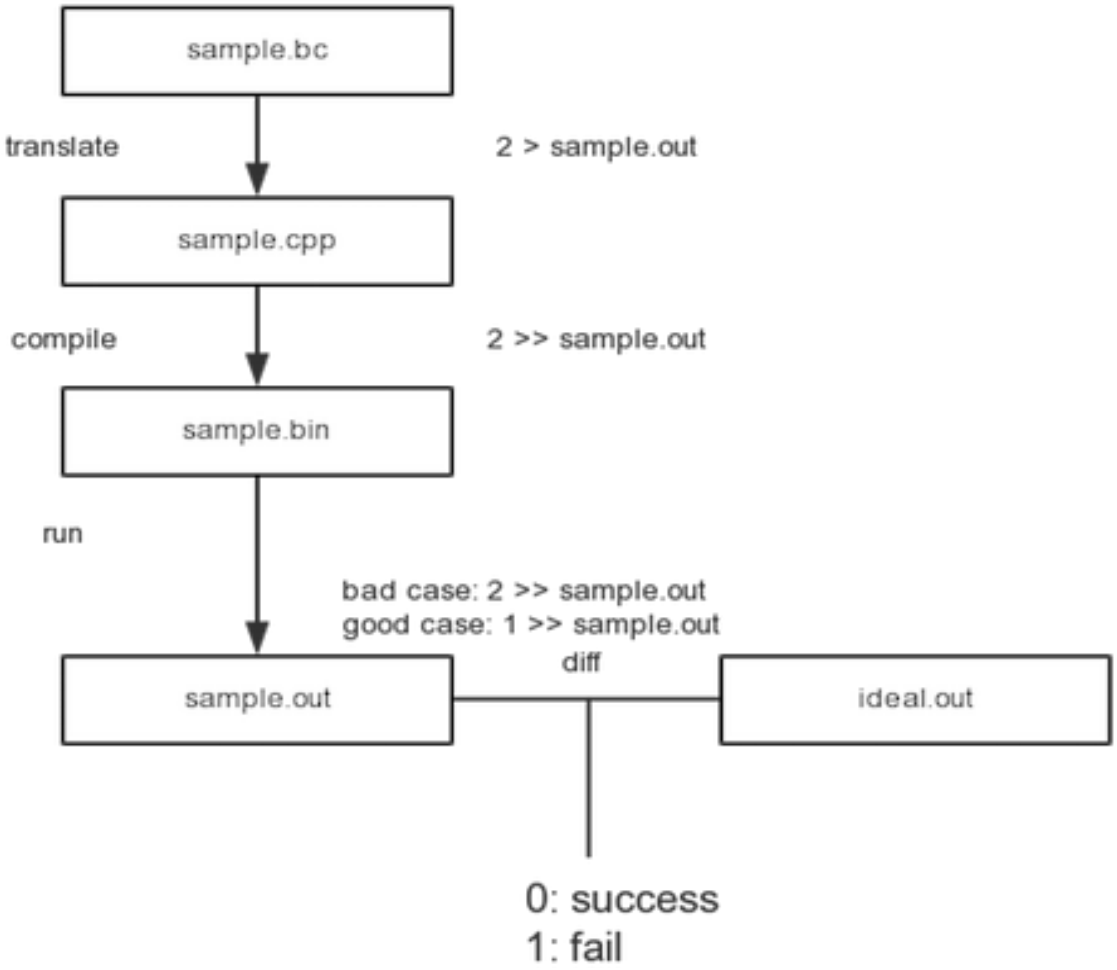
Chapter 6

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.

Appendix

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

Appendix A - BuckCal Library

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

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

void setrow(int_mat mat, int r, double_mat set); void setrow(string_mat mat, int r, string_mat set);	int mat and double mat can be mixed for 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.

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.

Appendix B - Code Listing

This appendix contains the code listing for BuckCal, including OCaml code, C++ implementation, BuckCal library and test cases. Exact line counts are not included for each test cases.

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

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	68
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	101

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

Summary: BuckCal Code

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

Summary: Test Cases

A total of 57 good cases and 73 bad cases are included. All good cases have 654 lines in total, and 411 lines for all bad cases.

There is another sample serves as a user programming example, which combines library functions and custom functions, two files of 118 lines total.

Summary: Scripts

File Name	Description	Lines of Code
test/test.sh	Automatic test script	120

usr/compile.sh	Easy-to-use scrpt for user to compile a program with full BuckCal library	56

All Source Code

scanner.mll

```
{
open Parser
open Lexing

exception Scanner_error of string

(* increase line no *)
let incr_lineno lexbuf =
  let pos = lexbuf.lex_curr_p in
  lexbuf.lex_curr_p <- { pos with
                         pos_lnum = pos.pos_lnum + 1;
                         pos_bol = pos.pos_cnum;
                       }
}

let upper = ['A'-'Z']
let lower = ['a'-'z']
let digit = ['0'-'9']

rule token = parse
       [' ' '\t' '\r'] { token lexbuf } (* Whitespace *)
     | '\n'    { incr_lineno lexbuf; token lexbuf } (* Newline *)
     | '#'     { comment lexbuf }      (* Comments *)
     | '('     { LPAREN }
     | ')'     { RPAREN }
     | '['     { LSBRACK }
     | ']'     { RSBRACK }
     | '{'     { LBRACE }
     | '}'     { RBRACE }
     | ';'     { SEMI }
     | ','     { COMMA }
     | '+'     { PLUS }
     | '-'     { MINUS }
     | '*'     { TIMES }
     | '/'     { DIVIDE }
     | ':'     { ASSIGN }
```

```
    | '='     { EQ }
    | "!="    { NEQ }
    | '<'     { LT }
    | "<="    { LEQ }
    | '>'     { GT }
    | ">="    { GEQ }
    | "not"   { NOT }
    | "and"   { AND }
    | "or"    { OR }
    | "if"    { IF }
    | "then"  { THEN }
    | "else"  { ELSE }
    | "elif"  { ELIF }
    | "fi"    { FI }
    | "for"   { FOR }
    | "in"    { IN }
    | "do"    { DO }
    | "rof"   { ROF}
    | "return" { RETURN }
    | "break"  { BREAK }
    | "continue" { CONTINUE }
    | "def"   { DEF }
    | "fed"   { FED }
    | "disp"  { DISP }
    | "int"   { INT }
    | "double" { DOUBLE }
    | "string" { STRING }
    | "bool"  { BOOL }
    | "true"  { BOOL_LITERAL(true) }
    | "false"  { BOOL_LITERAL(false) }
    | "int mat"      { INTMAT }
    | "double mat"   { DOUBLEMAT }
    | "string mat"   { STRINGMAT }
    | "import"       { IMPORT }
    | lower(lower|digit|'_')* as lxm { ID(lxm) }
    | digit+ as lxm { INT_LITERAL(int_of_string lxm) }
    | digit+'.'digit* as lxm {
      DOUBLE_LITERAL(float_of_string lxm) }
    | '\''    { let buffer = [] in
                STRING_LITERAL(string_lit buffer lexbuf) }
    | eof     { EOF }
    | _ as c  { let p = lexeme_start_p lexbuf in
                let msg = Printf.sprintf
                   "illegal character %s, in %s line %d,%d"
                   (Char.escaped c)
```

```
                p.pos_fname
                p.pos_lnum
                (p.pos_cnum - p.pos_bol + 1)
              in
              raise (Scanner_error msg) }

and comment = parse
   '\n' { incr_lineno lexbuf; token lexbuf }
  | eof  { EOF }
  | _    { comment lexbuf }

and string_lit buf = parse
   '\''     { String.concat "" (List.rev buf) }
  | eof     { raise (Scanner_error "Unexpected End-of-File") }
  | '\n'    { raise (Scanner_error "Unexpected End-of-Line") }
  | "\\n"    { string_lit ("\\n"::buf) lexbuf }
  | "\""    { string_lit ("\\\""::buf) lexbuf }
  | "\\t"    { string_lit ("\\t"::buf) lexbuf }
  | "\\'"    { string_lit ("\\'"::buf) lexbuf }
  | "\\\\"   { string_lit ("\\\\"::buf) lexbuf }
  | _ as c   { string_lit ((Char.escaped c)::buf) lexbuf }
```

parser.mly

```
%{
  open Ast
%}

%token LPAREN RPAREN LSBRACK RSBRACK LBRACE RBRACE SEMI COMMA ASSIGN
%token PLUS MINUS TIMES DIVIDE
%token IF THEN ELIF ELSE FI FOR IN DO ROF RETURN BREAK CONTINUE DEF FED
DISP
%token EQ NEQ LT LEQ GT GEQ NOT AND OR
%token INT DOUBLE STRING BOOL
%token INTMAT DOUBLEMAT STRINGMAT
%token IMPORT
%token <string> ID
%token <bool>  BOOL_LITERAL
%token <int> INT_LITERAL
%token <float> DOUBLE_LITERAL
%token <string> STRING_LITERAL
%token EOF

%right ASSIGN
%left OR
%left AND
```

```
%right NOT
%left LT GT LEQ GEQ
%left EQ NEQ
%left PLUS MINUS
%left TIMES DIVIDE
%nonassoc NEG

%start program
%type <Ast.program> program

%%

/* --- data type --- */

dt:
  | INT             { Int }
  | DOUBLE          { Double }
  | STRING          { String }
  | BOOL            { Bool }
  | INTMAT          { IntMat }
  | DOUBLEMAT       { DoubleMat }
  | STRINGMAT       { StringMat }

/* import */
import_stmt: IMPORT STRING_LITERAL { $2 }

import_stmts:               { [] }
  | import_stmts import_stmt    { $1 @ [$2] }

/* variable declaration or definition */
var_dec_def:
    dt ID SEMI              { VarNoInit({ vname = $2; vtype = $1; }) }
  | dt ID ASSIGN expr SEMI    { VarInit({ vname = $2; vtype = $1; }, $4) }


/* variable declaration or definition list */
var_dec_def_list:
    /* empty */             { [] }
  | var_dec_def_list var_dec_def  { $1 @ [$2] }


/* --- arguments and function related --- */

/* argument list in function declaration/definition */
arg_def_list_1 :
```

```
  | dt ID                  { [{ vname = $2; vtype = $1; }] }
  | dt                   { [{ vname = ""; vtype = $1; }] }
  | arg_def_list COMMA dt ID   { $1 @ [{ vname = $4; vtype = $3; }] }
  | arg_def_list COMMA dt      { $1 @ [{ vname = ""; vtype = $3; }] }

arg_def_list:
    /* empty */    { [] }
  | arg_def_list_1 { $1 }

/* a function declaration/definition */
func_def:
    DEF dt ID LPAREN arg_def_list RPAREN DO var_dec_def_list stmt_list FED {
      { return = $2; fname = $3; args = $5; locals = $8; body = $9; }
     }
  | DEF ID LPAREN arg_def_list RPAREN DO var_dec_def_list stmt_list FED {
       { return = Void; fname = $2; args = $4; locals = $7; body = $8; }
     }
  | DEF dt ID LPAREN arg_def_list RPAREN SEMI {
       { return = $2; fname = $3; args = $5; locals = []; body = []; }
     }
  | DEF ID LPAREN arg_def_list RPAREN SEMI {
       { return = Void; fname = $2; args = $4; locals = []; body = []; }
     }

/* a list of function definitions */
func_def_list:
    func_def             { [$1] } /* nothing */
  | func_def_list func_def  { $1 @ [$2] }


/* --- if related --- */

/* represent a series of elif */
elif_list:
    /* nothing */                  { [] }
  | elif_list ELIF expr THEN stmt_list   { $1 @ [{cond=$3; stmts=$5}] }


/* represent the optional else statement */
else_stmt:
    /* nothing */    { [] }
  | ELSE stmt_list    { $2 }


/* --- matrix literal related --- */
```

```
/* matrix literal, list of list */
mat_literal:
  | LBRACE RBRACE              { [[]] }
  | LBRACE mat_rows RBRACE     { $2 }

/* rows of matrix */
mat_rows:
  | mat_row              { [$1] } /* first row */
  | mat_rows SEMI mat_row  { $1 @ [$3] }

/* elements in a matrix row */
mat_row:
  | expr             { [$1] } /* first element */
  | mat_row COMMA expr  { $1 @ [$3]  }


left_value:
    ID                          { Id($1) }
  | ID LSBRACK expr COMMA expr RSBRACK /* matrix select */
                              { MatSub($1, $3, $5) }
/* an expression */
expr:
    BOOL_LITERAL                 { Boolval($1) }
  | INT_LITERAL                 { Intval($1) }
  | DOUBLE_LITERAL               { Doubleval($1) }
  | STRING_LITERAL               { Stringval($1) }
  | mat_literal                { Matval($1) }
  | left_value                 { Lvalue($1) }
  | expr PLUS   expr             { Binop($1, Plus,  $3) }
  | expr MINUS  expr             { Binop($1, Minus,  $3) }
  | expr TIMES  expr             { Binop($1, Times, $3) }
  | expr DIVIDE expr             { Binop($1, Divide,  $3) }
  | expr EQ     expr            { Binop($1, Eq,   $3) }
  | expr NEQ    expr             { Binop($1, Neq,  $3) }
  | expr LT     expr            { Binop($1, Lt,   $3) }
  | expr LEQ    expr             { Binop($1, Leq,  $3) }
  | expr GT     expr            { Binop($1, Gt,   $3) }
  | expr GEQ    expr             { Binop($1, Geq,  $3) }
  | expr AND    expr             { Binop($1, And,  $3) }
  | expr OR     expr            { Binop($1, Or,   $3) }
  | left_value ASSIGN expr          { Assign($1, $3) }
  | NOT expr                   { Unaop(Not, $2) }
  | MINUS expr %prec NEG           { Unaop(Neg, $2) }
  | ID LPAREN RPAREN              { Call($1, []) }
```

```
  | ID LPAREN arg_call_list RPAREN     { Call($1, $3) }
  | LPAREN expr RPAREN               { $2 }

/* argument list in function calling */
arg_call_list:
   expr        { [$1] }
  | arg_call_list COMMA expr  { $1 @ [ $3 ] }

/* --- program related --- */

/* a list of statements */
stmt_list:
   /* nothing */    { [] }
  | stmt_list stmt   { $1 @ [$2] }

/* a statement */
stmt:
   SEMI                              { Empty }
  | expr SEMI                           { Expr($1) }
  | RETURN expr SEMI                          { Return($2) }
  | IF expr THEN stmt_list elif_list else_stmt FI  { If(
                                    {cond = $2; stmts = $4},
                                    $5, $6) }
  | FOR ID IN expr DO stmt_list ROF             { CntFor($2, $4, $6) }
  | FOR expr DO stmt_list ROF                  { CndFor(
                                  {cond = $2; stmts = $4}) }
  | DISP expr SEMI                        { Disp($2) }
  | CONTINUE SEMI                           { Continue }
  | BREAK SEMI                             { Break }


/* a source file */
program:
   import_stmts func_def_list var_dec_def_list stmt_list  {
     {pimps = $1; pfuns = $2; pvars = $3; pstms = $4;}
    }
  | import_stmts var_dec_def_list stmt_list  {
      {pimps = $1; pfuns = []; pvars = $2; pstms = $3;}
    }
```

ast.ml

```
(* Abstract Syntax Tree *)
```

```
exception Syntax_error of string

(* operators *)
type binop =  Plus | Minus | Times | Divide
        | Eq | Neq | Lt | Leq | Gt | Geq | And | Or

type unaop = Not | Neg

(* data type *)
type dtype = Void | Int | Double | String | Bool | IntMat | DoubleMat | StringMat

(* variable definition *)
type var = { vtype: dtype; vname: string }
(* type var = Vint of string | Vdouble of string
       | Vstring of string | Vbool of string
       | Vintmat of string | Vdoublemat of string | Vstringmat of string *)

(* containers for the occasion that several type can work *)
(* type matelem_container = Int | Double | String *)
(* type matsub_container = Int | Double | String | Mat *)
(* type return_container = Void | Int | Double *)


(* expression *)
type lvalue =
   Id of string
  | MatSub of string * expr * expr
and expr =
   Boolval of bool
  | Intval of int
  | Doubleval of float
  | Stringval of string
  | Matval of expr list list
  | Lvalue of lvalue
  | Binop of expr * binop * expr
  | Assign of lvalue * expr
  | Unaop of unaop * expr
  | Call of string * expr list

(* variable declaration *)
type var_dec =
   VarNoInit of var
  | VarInit of var * expr

(* statement *)
```

```
type cond_stmts = {
  cond : expr ;
  stmts : stmt list;
}
and stmt =
    Expr of expr
  | Empty
  | Return of expr
  | If of cond_stmts * cond_stmts list * stmt list
  | CntFor of string * expr * stmt list
  | CndFor of cond_stmts
  | Disp of expr
  | Continue
  | Break

(* function definition *)
type func_def = {
  return : dtype;
  fname : string;
  args : var list;
  locals : var_dec list;
  body : stmt list;
}

(* import statement *)
(*type imp = string*)

(* program is function definition plus variable definition and statements *)
type program = {
  pimps : string list;
  pfuns : func_def list;
  pvars : var_dec list;
  pstms : stmt list;
}

(* A helper function: convert dtype to string *)
let pt t = match t with
    Int -> "Int"
  | Double -> "Double"
  | String -> "String"
  | Bool -> "Bool"
  | IntMat -> "IntMat"
  | DoubleMat -> "DoubleMat"
  | StringMat -> "StringMat"
  | Void -> "Void"
```

scheck.ml

```
(* Static Semantic Check
   Input AST, output SAST *)
open Ast
open Sast
open Scheck_expr
open Lib
open Printf

(* variable table *)
(* type var_table = svar_def list *)

(* find the type of a @name in @var_table
   return: (true, dtype) on found, (false, _) on not_found *)
let find_var var_table name =
  (*let _ = ignore(List.iter (fun (_,n,_) -> printf "%s " n) var_table) in
    let _ = printf "find: %s \n" name in*)
  try
    let t, _, _ = List.find (fun (_,b,_) -> b = name) var_table in
    true, t
  with Not_found -> false, Void

(* general type equality - int = double *)
let eq_t t1 t2 = match t1, t2 with
    Int, Double | Double, Int -> true
  | IntMat, DoubleMat | DoubleMat, IntMat -> true
  (*| Int, IntMat | Int, DoubleMat | Double, IntMat | Double, DoubleMat -> true*)
  | x, y -> if x=y then true else false


(* function table *)
(* type func_table = sfun_def list *)

(* helper: generate Sast.funsg of Sast.sfun_def *)
let sig_sfunc sfn = {
  fsname = sfn.sfname;
  fsargs = List.map (fun v -> v.vtype) sfn.sargs
}

(* helper: check if a sfun_def is declaration only *)
let is_func_dec ff = (ff.sbody=[] && ff.slocals=[])

(* helper: print function signature *)
```

```
let print_func_sig sfn =
  let rec args_s xx = match xx with
      [] -> ""
    | [a] -> pt a
    | a::b::tl -> (sprintf "%s, %s" (pt a) (args_s (b::tl)))
  in
  eprintf "%s(%s)\n" sfn.fsname (args_s sfn.fsargs)

(* replace @o in List @lst with @n *)
let rec list_rep o n lst = match lst with
    [] -> []
  | hd::tl -> let a = (if hd=o then n else hd) in a::(list_rep o n tl)

(* find a function signature in function table
   arguments: @eq - the equal operator : can be (=) or eq_t
              @fnsg - function signature to be found
              @func_table - function table
   return: (true, sfun_def) on found, (false, _) on not_found *)
let find_func eq func_table fnsg =
  let dummy = {sreturn=Void; sfname="_"; sargs=[]; slocals=[]; sbody=[]} in
  let func_eq f1 fd =
    let f2 = sig_sfunc fd in
    f1.fsname = f2.fsname &&
    try List.for_all2 eq f1.fsargs f2.fsargs
    with Invalid_argument _ -> false
  in
  try true, (List.find (func_eq fnsg) func_table)
  with Not_found -> false, dummy


(* variable default value,
   return a sexpression *)
let svar_init_sexpr var = match var with
    Int -> Int, SIntval 0
  | Double -> Double, SDoubleval 0.0
  | Bool -> Bool, SBoolval false
  | String -> String, SStringval ""
  | IntMat -> IntMat, SMatval ([[]], 0, 0)
  | DoubleMat -> DoubleMat, SMatval ([[]], 0, 0)
  | StringMat -> StringMat, SMatval ([[]], 0, 0)
  | Void -> raise (Bad_type "cannot define a void variable")

(* convert var list to svar_def list *)
let var2def_list vl =
  let var2def v = v.vtype, v.vname, (svar_init_sexpr v.vtype) in
```

```
  List.map var2def vl

(* check expr,
   return a sexpression *)
let rec check_lvalue ftbl vtbl lv = match lv with
    Id x -> let f, t = find_var vtbl x in
    if f then t, (SId x)
    else raise (Bad_type ("variable " ^ x ^ " not defined"))
  | MatSub(x, e1, e2) -> let f, t = find_var vtbl x in
    if f then begin
     let new_t = match t with
                 IntMat -> Int
               | DoubleMat -> Double
               | StringMat -> String
               | _ -> raise (Bad_type ("bad matsub operator"))
     in
     let te1, se1 = check_expr ftbl vtbl e1 in
     let te2, se2 = check_expr ftbl vtbl e2 in
     if te1 = Int && te2 = Int then
      (new_t, SMatSub (x, (te1, se1), (te2, se2)))
     else raise (Bad_type ("Submat index must be int"))
    end
    else raise (Bad_type ("variable " ^ x ^ " not defined"))
and check_matval ftbl vtbl matx =
  let check_exp_list exp_list_list =
    List.map (List.map (check_expr ftbl vtbl)) exp_list_list
  in
  check_matval_s (check_exp_list matx)
and check_call ftbl vtbl fn exp_list =
  let sexp_list = List.map (check_expr ftbl vtbl) exp_list in
  let typ_list = List.map (fst) sexp_list in
  let found, fnsg = find_func eq_t ftbl {fsname=fn; fsargs=typ_list} in
  if found && not (is_func_dec fnsg) then fnsg.sreturn, SCall(fn, sexp_list)
  else raise (Bad_type ("function " ^ fn ^ " not defined"))
and check_expr ftbl vtbl exp = match exp with
    Intval x -> Int, SIntval x
  | Doubleval x -> Double, SDoubleval x
  | Stringval x -> String, SStringval x
  | Boolval x -> Bool, SBoolval x
  | Matval matx -> check_matval ftbl vtbl matx
  | Lvalue lv -> check_lvalue ftbl vtbl lv
  | Binop(e1, bop, e2) -> check_binop bop
                          (check_expr ftbl vtbl e1)
                          (check_expr ftbl vtbl e2)
  | Unaop(uop, x) -> check_uniop uop (check_expr ftbl vtbl x)
```

```
  | Assign(lv, x) -> check_assign (check_lvalue ftbl vtbl lv)
                     (check_expr ftbl vtbl x)
  | Call(fn, xl) -> check_call ftbl vtbl fn xl


(* check variable definition list,
   while building variable table
   return a svar_def list *)
let rec check_vardecs ftbl vtbl vardecs = match vardecs with
    [] -> vtbl
  | hd::tl -> let new_vardec = (
    let new_v, init_e = (
      match hd with
        VarNoInit v -> v, (svar_init_sexpr v.vtype)
      | VarInit (v, e) -> v, (check_expr ftbl vtbl e))
    in
    let new_type, new_name = new_v.vtype, new_v.vname in
    let _ =
      let f, _ = find_var vtbl new_name in
      if not f then () else raise (Bad_type (new_v.vname ^ " defined twice"))
    in
    let new_type =
      if eq_t new_type (fst init_e) then new_type
      else raise (Bad_type "variable and expression type mismatch")
    in
    [(new_type, new_name, init_e)] )
    in
    (check_vardecs ftbl (vtbl@new_vardec) tl)

(* get svar locals from var list, check all var with incremental vtbl, but not returning a
merged vtbl *)
let rec check_local_var_def ftbl vtbl local_var_list = match local_var_list with
    [] -> []
  | hd::tl -> let new_vardec = (
    let new_v, init_e = (
      match hd with
        VarNoInit v -> v, (svar_init_sexpr v.vtype)
      | VarInit (v, e) -> v, (check_expr ftbl vtbl e))
    in
    let new_type, new_name = new_v.vtype, new_v.vname in
    let _ =
      let f, _ = find_var vtbl new_name in
      if not f then () else raise (Bad_type (new_v.vname ^ " defined twice"))
    in
    let new_type =
```

```
    if eq_t new_type (fst init_e) then new_type
    else raise (Bad_type "variable and expression type mismatch")
  in
  [(new_type, new_name, init_e)] )
  in
  (new_vardec @ (check_local_var_def ftbl (new_vardec@vtbl) tl))


(* check statement list.
  return: sstmt list *)
let rec check_condstmts ftbl vtbl ret_type loop_flag cs = match cs with(* translate a list of
elif *)
   [] -> []
 | hd::tl -> let _, sstmts = (check_stmts ftbl vtbl ret_type false false loop_flag hd.stmts) in
   { scond = (check_expr ftbl vtbl hd.cond) ;
     sstmts
   } :: (check_condstmts ftbl vtbl ret_type loop_flag tl )
and check_stmts ftbl vtbl ret_type main_flag ret_flag loop_flag stmts= match stmts with
   [] -> ret_flag,[]
 | hd::tl ->
   let flag0, flist0 =
     ( match hd with
         Empty -> ret_flag, SEmpty
       | Expr e -> ret_flag, SExpr (check_expr ftbl vtbl e)
       | Return e ->
         let ret = check_expr ftbl vtbl e in
         if fst ret == ret_type
         then (if (main_flag) then true,SReturn ret else false,SReturn ret)
         else raise (Bad_type "mismatch with function's return type")
       | If (c, cl, ss) -> let _, check_ss = check_stmts ftbl vtbl ret_type false ret_flag loop_flag ss
in
         ret_flag, SIf ((List.hd (check_condstmts ftbl vtbl ret_type loop_flag [c] )),
                    (check_condstmts ftbl vtbl ret_type loop_flag cl),
                    check_ss
                   )
       | CntFor (s, e, ss) -> (
           let f, st = find_var vtbl s in
           let et, e = check_expr ftbl vtbl e in
           let _, sss = check_stmts ftbl vtbl ret_type false ret_flag true ss  in
           let _ = if f then () else raise (Bad_type (s ^ "undefined")) in
           let et_t = match et with
              IntMat -> Int
            | DoubleMat -> Double
            | StringMat -> String
            | _ -> raise (Bad_type "must be loop in a mat")
```

```
          in
          if eq_t et_t st then ret_flag, SCntFor (s, (et, e), sss)
          else raise (Bad_type "loop variable type mismatch") )
       | CndFor cs -> ret_flag,SCndFor (List.hd (check_condstmts ftbl vtbl ret_type true
[cs]))
       | Disp e -> ret_flag, SDisp (check_expr ftbl vtbl e)
       | Continue -> if(loop_flag) then ret_flag, SContinue else raise (Bad_type "Continue
should only be used inside a loop")
       | Break -> if(loop_flag) then ret_flag, SBreak else raise (Bad_type "Break should only
be used inside a loop")
     ) in
   let flag1, flist1 = check_stmts ftbl vtbl ret_type main_flag ret_flag loop_flag tl
   in flag0||flag1 , flist0::flist1


(* check_fundef
    check function definition
  arguments: Sast.sfun_def list, Ast.func_def
  return: Sast.sfun_def list *)
let check_fundef new_ftbl ftbl new_func_def =
 let sig_func fn = {
  fsname = fn.fname;
  fsargs = List.map (fun v -> v.vtype) fn.args
 } in
 let new_fnsg = sig_func new_func_def in (* signature *)
 let new_sret = new_func_def.return in (* return type *)
 (*let _ = print_func_sig new_fnsg in*)
 let new_sname = new_func_def.fname in (* name *)
 let new_sargs = new_func_def.args in (* arguments *)
 (* check local variables & build variable table *)
 let arg_def = var2def_list new_sargs in
 let full_ftbl = ftbl @ new_ftbl in
 let new_local = check_local_var_def full_ftbl arg_def new_func_def.locals in
 let vtbl = (arg_def@new_local) in
 (* check statements *)
 let flag, new_fstmts = check_stmts full_ftbl vtbl new_sret true false false
new_func_def.body in
 let new_sfun_def = { sreturn = new_sret;
                      sfname = new_sname;
                      sargs = new_sargs;
                      slocals = new_local;
                      sbody = new_fstmts } in
 let _ = if (new_sret != Void && not (is_func_dec new_sfun_def) && not flag)
  then raise (Bad_type ("Function '" ^ new_sname ^ "' return statement missing"))
  else ()
```

```
  in
  let found, _ = find_func (=) (ftbl) new_fnsg in
  let foundnew, fbodynew = find_func (=) (new_ftbl) new_fnsg in
  (*let _ = eprintf "%s: %s" new_sname (if found then "found" else "not found") in*)
  match found, foundnew with
    false, false -> new_ftbl @ [new_sfun_def]
  | false, true ->  if (is_func_dec fbodynew) && not (is_func_dec new_sfun_def)
                then begin
                  if fbodynew.sreturn = new_sret then
                    (list_rep fbodynew new_sfun_def new_ftbl)
                  else
                    raise (Bad_type ("Function '" ^ new_sname ^ "' return type different with
declaration"))
                end
                else raise (Bad_type ("Function '" ^ new_sname ^ "' already defined"))
  | true, _ -> raise (Bad_type ("Function '" ^ new_sname ^ "' already defined"))


(* check function definition list
   input: func_def list
   return: sfun_def list *)
let rec check_fundefs new_ftbl ftbl funsgs = match funsgs with
    [] -> new_ftbl
  | hd::tl -> let new_ftbl = check_fundef new_ftbl ftbl hd in
              check_fundefs new_ftbl ftbl tl


(* check the whole program
   returns: sprogram
   note: lib_funs is imported by default
 *)
let check need_dec_extern extern_funs prg =
  let func_table =
    let func_table_0 = lib_funs @ extern_funs in (* init function table (should be built-in
functions)
                                and init new function table (user-defined & empty) *)
    check_fundefs [] func_table_0 prg.pfuns
  in
  let full_ftbl =  lib_funs @ extern_funs @ func_table in
  let var_table =
    let var_table_0 = [] in    (* init variable table as empty *)
    check_vardecs full_ftbl var_table_0 prg.pvars
  in
  let _, stm_lines =           (* statements *)
    check_stmts full_ftbl var_table Int true true false prg.pstms
```

```
  in
  match need_dec_extern with
    IMP -> { spfuns = func_table; spvars = []; spstms = [] }
  | _ -> { spfuns = func_table; spvars = var_table; spstms = stm_lines }
```

scheck_expr.ml

```
(* sub-routines called in shcek_expr in scheck.ml *)
open Ast
open Sast
open Printf


let check_uniop uop sexp =
  let ret = SUnaop(uop, sexp) in
  match uop with
    Not -> (match sexp with
        Bool, _ -> Bool, ret
      | _, _ -> raise (Bad_type "\"not\" bad operand type"))
  | Neg -> (match sexp with
        Int, _ -> Int, ret
      | Double, _ -> Double, ret
      | _, _ -> raise (Bad_type "unary negitive: bad operand type"))


let check_binop bop sexp1 sexp2 =
  let t1, _ = sexp1 in
  let t2, _ = sexp2 in
  let ret0 = SBinop(sexp1, bop, sexp2) in
  match bop with
    Plus -> (match t1, t2 with
      (* scalar arithmetic binary op *)
        Int, Int -> Int, ret0
      | Double, Double -> Double, ret0
      | Int, Double -> Double, ret0
      | Double, Int -> Double, ret0
      | String, String -> String, ret0
      (* matrix arithmetic binary op *)
      | IntMat, IntMat -> IntMat, ret0
      | DoubleMat, IntMat -> DoubleMat, ret0
      | IntMat, DoubleMat -> DoubleMat, ret0
      | DoubleMat, DoubleMat -> DoubleMat, ret0
      | StringMat, StringMat -> StringMat, ret0
      (* matrix-scalar arithmetic binary op *)
      | IntMat, Int -> IntMat, ret0
```

```
    | DoubleMat, Int -> DoubleMat, ret0
    | IntMat, Double -> DoubleMat, ret0
    | _, _ -> raise (Bad_type "\"+\" bad operand type"))
  | Minus -> (match t1, t2 with
    (* scalar arithmetic binary op *)
      Int, Int -> Int, ret0
    | Double, Double -> Double, ret0
    | Int, Double -> Double, ret0
    | Double, Int -> Double, ret0
    (* matrix arithmetic binary op *)
    | IntMat, IntMat -> IntMat, ret0
    | DoubleMat, IntMat -> DoubleMat, ret0
    | IntMat, DoubleMat -> DoubleMat, ret0
    | DoubleMat, DoubleMat -> DoubleMat, ret0
    (* matrix-scalar arithmetic binary op *)
    | IntMat, Int -> IntMat, ret0
    | DoubleMat, Int -> DoubleMat, ret0
    | IntMat, Double -> DoubleMat, ret0
    | _, _ -> raise (Bad_type "\"-\" bad operand type"))
  | Times -> (match t1, t2 with
    (* scalar arithmetic binary op *)
      Int, Int -> Int, ret0
    | Double, Double -> Double, ret0
    | Int, Double -> Double, ret0
    | Double, Int -> Double, ret0
    (* matrix arithmetic binary op *)
    | IntMat, IntMat -> IntMat, ret0
    | DoubleMat, IntMat -> DoubleMat, ret0
    | IntMat, DoubleMat -> DoubleMat, ret0
    | DoubleMat, DoubleMat -> DoubleMat, ret0
    (* matrix-scalar arithmetic binary op *)
    | IntMat, Int -> IntMat, ret0
    | DoubleMat, Int -> DoubleMat, ret0
    | IntMat, Double -> DoubleMat, ret0
    | _, _ -> raise (Bad_type "\"*\" bad operand type"))
  | Divide -> (match t1, t2 with
    (* scalar arithmetic binary op *)
      Int, Int -> Int, ret0
    | Double, Double -> Double, ret0
    | Int, Double -> Double, ret0
    | Double, Int -> Double, ret0
    (* matrix arithmetic binary op *)
    | IntMat, IntMat -> IntMat, ret0
    | DoubleMat, IntMat -> DoubleMat, ret0
    | IntMat, DoubleMat -> DoubleMat, ret0
```

```
    | DoubleMat, DoubleMat -> DoubleMat, ret0
    (* matrix-scalar arithmetic binary op *)
    | IntMat, Int -> IntMat, ret0
    | DoubleMat, Int -> DoubleMat, ret0
    | IntMat, Double -> DoubleMat, ret0
    | _, _ -> raise (Bad_type "\"/\" bad operand type"))
  | Eq -> (match t1, t2 with
      Int, Int -> Bool, ret0
    | Double, Double -> Bool, ret0
    | Int, Double -> Bool, ret0
    | Double, Int -> Bool, ret0
    | String, String -> Bool, ret0
    | Bool, Bool -> Bool, ret0
    | _, _ -> raise (Bad_type "\"=\" bad operand type"))
  | Neq -> (match t1, t2 with
      Int, Int -> Bool, ret0
    | Double, Double -> Bool, ret0
    | Int, Double -> Bool, ret0
    | Double, Int -> Bool, ret0
    | String, String -> Bool, ret0
    | _, _ -> raise (Bad_type "\"!=\" bad operand type"))
  | Lt -> (match t1, t2 with
      Int, Int -> Bool, ret0
    | Double, Double -> Bool, ret0
    | Int, Double -> Bool, ret0
    | Double, Int -> Bool, ret0
    | String, String -> Bool, ret0
    | _, _ -> raise (Bad_type "\"<\" bad operand type"))
  | Leq -> (match t1, t2 with
      Int, Int -> Bool, ret0
    | Double, Double -> Bool, ret0
    | Int, Double -> Bool, ret0
    | Double, Int -> Bool, ret0
    | String, String -> Bool, ret0
    | _, _ -> raise (Bad_type "\"<=\" bad operand type"))
  | Gt -> (match t1, t2 with
      Int, Int -> Bool, ret0
    | Double, Double -> Bool, ret0
    | Int, Double -> Bool, ret0
    | Double, Int -> Bool, ret0
    | String, String -> Bool, ret0
    | _, _ -> raise (Bad_type "\">\" bad operand type"))
  | Geq -> (match t1, t2 with
      Int, Int -> Bool, ret0
    | Double, Double -> Bool, ret0
```

```
    | Int, Double -> Bool, ret0
    | Double, Int -> Bool, ret0
    | String, String -> Bool, ret0
    | _, _ -> raise (Bad_type "\">=\" bad operand type"))
  | And -> (match t1, t2 with
      Bool, Bool -> Bool, ret0
    | _, _ -> raise (Bad_type "\"and\" bad operand type"))
  | Or -> (match t1, t2 with
      Bool, Bool -> Bool, ret0
    | _, _ -> raise (Bad_type "\"or\" bad operand type"))


let check_matval_s sexp_list_list =
  let size_check tll =
    let ncol_list = List.map (List.length) tll in (* ncol of each row *)
    let num_col = (* get number of columns while checking *)
      let helpr a b = match a, b with
        (-1), y -> y
      | x, y -> (if x = y then y
              else raise (Bad_type "Mat rows must have same length"))
      in List.fold_left helpr (-1) ncol_list in
    let num_row = List.length ncol_list in
    (num_row, num_col)
  in
  let type_check tll =
    (*let mat_elem_type = [Int; Double; String] in*)
    let helpr a b = match a, b with
      _, Void -> raise (Bad_type "Mat elements cannot be void")
    | Void, y -> y
    | Int, Int -> Int
    | Int, Double -> Double
    | Double, Double -> Double
    | Double, Int -> Double
    | String, String -> String
    | x, y -> raise (Bad_type (sprintf "Mat elements must have same type (%s, %s)" (pt x)
(pt y)))
    in
    List.fold_left helpr Void (List.flatten tll)
  in
  let typ_ll = List.map (List.map fst) sexp_list_list in
  let ncol, nrow = size_check typ_ll in
  let rt = (match (type_check typ_ll) with
      Int -> IntMat
    | Double -> DoubleMat
    | String -> StringMat
```

```
    | _ -> raise (Bad_type "Mat can only contain int, double or string")
   ) in
  rt, (SMatval (sexp_list_list, ncol, nrow))


let check_assign sexp1 sexp2 =
  let t1, _ = sexp1 in
  let t2, _ = sexp2 in
  let ret0 = SAssign(sexp1, sexp2) in
  match t1, t2 with
  (* scalar assignment *)
    Int, Int -> Int, ret0
  | Double, Int -> Double, ret0
  | Double, Double -> Double, ret0
  | Int, Double -> Double, ret0
  | String, String -> String, ret0
  | Bool, Bool -> Bool, ret0
  (* matrix assignment *)
  | IntMat, IntMat -> IntMat, ret0
  | IntMat, DoubleMat -> IntMat, ret0
  | DoubleMat, DoubleMat -> DoubleMat, ret0
  | DoubleMat, IntMat -> DoubleMat, ret0
  | StringMat, StringMat -> StringMat, ret0
  (* 1-by-1 mat assigned <-> scalar *)
  | Int, IntMat -> Int, ret0
  | Double, IntMat -> Double, ret0
  | Int, DoubleMat -> Int, ret0
  | Double, DoubleMat -> Double, ret0
  | String, StringMat -> String, ret0
  | IntMat, Int -> Int, ret0
  | IntMat, Double -> Double, ret0
  | DoubleMat, Int -> Int, ret0
  | DoubleMat, Double -> Double, ret0
  | StringMat, String -> String, ret0
  | x, y -> raise (Bad_type (sprintf "%s : %s operand types invalid" (pt x) (pt y)))
```

sast.ml

```
(* Semantic checked Abstract Syntax Tree - Safe Abstract Syntax Tree *)
open Ast

(* Is this top level file ? Or
   Is main() needed ? *)
type top_level = TOP | IMP
```

```
exception Bad_type of string

(* variable *)
(* type svar = { svtype: dtype; svname: string } *)

(* expression with type *)
type sexpr_val =
   SId of string
  | SMatSub of string * sexpr * sexpr
  | SBoolval of bool
  | SIntval of int
  | SDoubleval of float
  | SStringval of string
  | SMatval of sexpr list list * int * int (* values, nrow, ncol *)
  | SBinop of sexpr * binop * sexpr
  | SAssign of sexpr * sexpr
  | SUnaop of unaop * sexpr
  | SCall of string * sexpr list
and sexpr = dtype * sexpr_val

(* variable definition *)
type svar_def = dtype * string * sexpr (* type, name, init expr *)

(* statement *)
type scond_stmts = {
  scond : sexpr;
  sstmts : sstmt list
}
and sstmt =
   SEmpty
  | SExpr of sexpr
  | SReturn of sexpr
  | SIf of scond_stmts * scond_stmts list * sstmt list
  | SCntFor of string * sexpr * sstmt list
  | SCndFor of scond_stmts
  | SDisp of sexpr
  | SContinue
  | SBreak

(* function signature *)
type funsg = {
  fsname : string;
  fsargs : dtype list
}
```

```
(* function definition *)
type sfun_def = {
  sreturn : dtype;
  sfname : string;
  sargs : var list;
  slocals : svar_def list;
  sbody : sstmt list
}


(* program *)
type sprogram = {
  spfuns : sfun_def list;
  spvars : svar_def list;
  spstms : sstmt list;
}
```

translate.ml

```
(*
  Translate SAST to TAST
  Input: SAST
  Output: TAST
*)

open Ast
open Sast
open Tast
open Translate_expr
open Printf


let gen_fname s = if s = "main" then "B_main" else s

(* translate expr.
   Note that translate an Matval may result in extra irstmt *)
(* @isl: irstmt list *)
(* return : irstmt list * irexpr *)
let rec trans_expr tid isl exp = match exp with
    _, SIntval x -> isl, (IIntval x)
  | _, SDoubleval x -> isl, (IDoubleval x)
  | _, SStringval x -> isl, (IStringval x)
  | _, SBoolval x -> isl, (IBoolval x)
  | _, SId x -> isl, (IId x)
  | _, SBinop (e1, b, e2) -> (let isl, ie1 = trans_expr tid isl e1 in
```

```
                    let isl, ie2 = trans_expr tid isl e2 in
                    let isl1, ret = trans_binop ie1 ie2 b in
                    (isl@isl1, ret))
  | _, SAssign (e1, e2) -> (let isl, ie1 = trans_expr tid isl e1 in
                  let isl, ie2 = trans_expr tid isl e2 in
                  (isl, (IAssign (ie1, ie2))))
  | _, SUnaop (u, e) -> let isl, ie = trans_expr tid isl e in
                (isl, (IUnaop (u, ie)))
  | _, SCall (s, el) -> let s = gen_fname s in
                let isl, iesl = trans_arglist tid isl el in
                (isl, ICall (s, iesl))
  | _, SMatSub (s, e1, e2) -> let isl, ie1 = trans_expr tid isl e1 in
                    let isl, ie2 = trans_expr tid isl e2 in
                    (trans_matsub s isl ie1 ie2)
  | t, SMatval (ell, nr, nc) -> (
     let arr = trans_matval ell in
     let ta = smat_to_array t in
     let tname = sprintf "T_%d_%d" tid (List.length isl) in
     let ttname = sprintf "TT_%d_%d" tid (List.length isl) in
     let isl = isl@[IVarDec (ta, tname, arr)] in
     let ex = ICall ((smat_to_cnsr t), [IId tname; IIntval nr; IIntval nc]) in
     let isl = isl@[IVarDec ((ipt t), ttname, ex)] in
     (isl, (IId ttname)))
and trans_arglist tid isl el = match el with
    [] -> isl, []
  | e::tl -> (let isl, ie = trans_expr tid isl e in
          let isl, itl = trans_arglist tid isl tl in
          isl, ie::itl)
and trans_matval ell = (* matrix element should not generate extra irstmt *)
  let el = List.flatten ell in
  let irel = List.map (fun x -> snd (trans_expr 0 [] x)) el in
  (IArray irel)

(* translate variable definition list *)
(* return tid, statement list *)
let rec trans_vardecs tid vars = match vars with
    [] -> tid, []
  | (t, s, e)::tl -> let hd_x = (
    let it = ipt t in
    let isl, ie = trans_expr tid [] e in
    (isl@[IVarDec (it, s, ie)])
  ) in
    let tid, tl_x = trans_vardecs (tid + 1) tl in
    ((tid + 1), (hd_x@tl_x))
```

```
(*
  translate statement.
  A temporary variable id (@tid) is for preventing naming conflict
  return : tid, statement list
*)
let rec trans_stmts tid stmts = (*print_int tid;*) match stmts with
    [] -> tid, []
  | hd::tl -> let tid, hd_stmts =
    ( match hd with
      | SEmpty -> (tid, [IEmpty])
      | SExpr e -> let isl, ie = trans_expr tid [] e in (tid, isl@[IExpr ie])
      | SReturn e -> let isl, ie = trans_expr tid [] e in (tid, isl@[IReturn ie])
      | SIf (cs, csl, sl) ->
        let tid, isl1, stmts1 =
          let isl0, ie, is = trans_condstmt tid [] cs in
          ((tid+1), isl0, ([IIfHead ie] @ is))
        in
        let tid, (isl2, stmts2) = (tid+1), trans_condstmts tid csl in
        let tid, part3 =
          let tid, is3 = trans_stmts tid sl in
          ((tid+1), ([IElse] @ is3))
        in
        (tid, ((isl1 @ isl2) @ (stmts1 @ stmts2) @ part3 @ [IBlockEnd]))
      | SCntFor (s, e, ss) ->
        let iv = ("F_" ^ s) in
        let fs1 = IVarDec(Iint, iv, (IIntval 0)) in
        let isl, temparr = (trans_expr tid [] e) in
        let tt = (sprintf "TT_%d" tid) in
        let tarrtype = ipt (fst e) in
        let tt_array = IVarDec(tarrtype, tt, temparr) in
        let fh = IForHead(fs1, (IBinop(IId iv, Lt, (IBinop((rows tt),Times,(cols tt))))),
                          (IAssign(IId iv, IBinop(IId iv, Plus, int1))))
        in
        let mainbody =
          let lbody_h = IExpr (IAssign(IId s, IIndex(tt, IId iv))) in
          let _, lbody = trans_stmts (tid + 1) ss in
          (lbody_h :: lbody)
        in
        ((tid+1), ( isl @ [tt_array] @ [fh] @ mainbody @ [IBlockEnd] ))
      | SCndFor cs -> let isl0, ie, is = trans_condstmt tid [] cs in
        (tid+1, (isl0 @ [IWhileHead ie] @ is @ [IBlockEnd]))
      | SDisp e -> let isl, ie = trans_expr tid [] e in (tid+1, isl@[IDisp ie])
      | SContinue -> (tid, [IContinue])
      | SBreak -> (tid, [IBreak])
    ) in
```

```
    let tid, tl_stmts = trans_stmts (tid + 1) tl in
    (tid, hd_stmts@tl_stmts)
and trans_condstmt tid isl cs =
  let isl0, iec = trans_expr tid isl cs.scond in
  let _, iss = trans_stmts (tid + 1) cs.sstmts in
  (isl0, iec, iss)
and trans_condstmts tid condstmtlist = match condstmtlist with
    [] -> [], []
  | hd::tl -> let isl1, stmts1 =
    let isl2, ie2, is2 = trans_condstmt tid [] hd in
    (isl2, ([IElseIf ie2] @ is2))
    in
    let isls, stmtss = trans_condstmts (tid+1) tl in
    (isl1@isls, stmts1@stmtss)


(* translate main function - add return 0 if no statment of the last one is not return *)
let trans_main_func tid stmts =
  let _, main_stmts = trans_stmts tid stmts in
  match (List.rev main_stmts) with
    [] -> [IReturn (IIntval 0)]
  | hd::_ -> match hd with IReturn _ -> main_stmts
                    | _ -> main_stmts @ [IReturn int0]

(* translate function declaration/definition *)
let rec trans_args args = match args with
    [] -> []
  | a::tl -> {ivtype = (ipt a.vtype); ivname = a.vname} :: (trans_args tl)

let rec trans_fundefs fundefs = match fundefs with
    [] -> []
  | hd::tl -> (
      let _, ss_v = trans_vardecs 0 hd.slocals in
      let _, ss_s = trans_stmts 0 hd.sbody in
      let fname = gen_fname hd.sfname in
      { ireturn = (ipt hd.sreturn); ifname = fname;
        iargs = (trans_args hd.sargs); ibody = (ss_v@ss_s) }
    )::(trans_fundefs tl)


(* translate whole program *)
let translate need_main prg =
  let func_lines =
    let funs = prg.spfuns in
    trans_fundefs funs
```

```
  in
  let tid, var_lines =
    let vars = prg.spvars in
    trans_vardecs 0 vars
  in
  let stmt_lines =
    let stmts = prg.spstms in
    trans_main_func tid stmts
  in
  let main_func = {
    ireturn = Iint;
    ifname = "main";
    iargs = [];
    ibody = [Itry] @ var_lines @ stmt_lines @ [ICatch]
  } in
  match need_main with
    TOP -> { ivars = []; ifuns = func_lines @ [main_func] }
  | _ -> { ivars = []; ifuns = func_lines }
```

translate_expr.ml

```
(* Helper functions in translate *)
open Ast
(*open Sast*)
open Tast

(* Translate dtype to C++ types *)
let ipt t = match t with
    Int -> Iint
  | Double -> Idouble
  | String -> Istring
  | Bool -> Ibool
  | IntMat -> Iint_mat
  | DoubleMat -> Idouble_mat
  | StringMat -> Istring_mat
  | Void -> Ivoid

(* Translate mat type to C++ array type *)
let smat_to_array m = match m with
    IntMat -> Iint_array
  | DoubleMat -> Idouble_array
  | StringMat -> Istring_array
  | _ -> raise (Not_now "Mat should be IntMat, DoubleMat, StringMat")

(* Translate mat type to constructor name *)
```

```
let smat_to_cnsr m = match m with
   IntMat -> "int_mat"
  | DoubleMat -> "double_mat"
  | StringMat -> "string_mat"
  | _ -> raise (Not_now "Mat should be IntMat, DoubleMat, StringMat")

(* Int 0 and 1 *)
let int0 = IIntval 0
let int1 = IIntval 1
(* rows and columns of a mat @s *)
let rows s = ICall("rows", [IId s])
let cols s = ICall("cols", [IId s])

let trans_binop e1 e2 b = [], IBinop (e1, b, e2)

let trans_matsub s isl ie1 ie2 =
  let r = rows s in
  let c = cols s in
  let x_ = IBinop (ie1, Minus, int1) in
  let y_ = IBinop (ie2, Minus, int1) in
  let check_x = IBinop(IBinop(x_, Geq, int0), And, IBinop(x_, Lt, r)) in
  let check_y = IBinop(IBinop(y_, Geq, int0), And, IBinop(y_, Lt, c)) in
  let assert_stmt = [ICheck ("matsub: index check failed, bad index", IBinop (check_x, And,
check_y))] in
  isl@assert_stmt, IIndex (s, IBinop(IBinop(x_, Times, c), Plus, y_))
```

tast.ml

```
(* target language AST *)
open Ast

exception Not_now of string

(* target data type *)
type itype = Ivoid | Iint | Idouble | Istring | Ibool
        | Iint_array | Idouble_array | Istring_array
        | Iint_mat | Idouble_mat | Istring_mat

type ivar = {
  ivtype : itype;
  ivname : string
}

(* target expression *)
type iexpr =
```

```
    IId of string
  (*| IMatSub of string * iexpr * iexpr * iexpr * iexpr (* M(x, y, r, c) *)*)
  | IIntval of int
  | IDoubleval of float
  | IStringval of string
  | IBoolval of bool
  | IArray of iexpr list
  | IBinop of iexpr * binop * iexpr
  | IAssign of iexpr * iexpr
  | IUnaop of unaop * iexpr
  | ICall of string * iexpr list
  | IIndex of string * iexpr

(* target variable declare *)
type ivar_dec = itype * string * iexpr (* type, name, init expr *)

(* target statement *)
type irstmt =
    IEmpty
  | IVarDec of ivar_dec
  | IExpr of iexpr
  | IReturn of iexpr (* return e *)
  | IIfHead of iexpr (* if (e) { *)
  | IElseIf of iexpr (* } else if (e) { *)
  | IElse (* } else { *)
  | IForHead of irstmt * iexpr * iexpr (* for (s; s; e) { *)
  | IWhileHead of iexpr (* while (e) { *)
  | IBlockEnd (* } *)
  | IDisp of iexpr (* cout << e << endl *)
  | IContinue (* continue *)
  | IBreak (* break *)
  | ICheck of string * iexpr (* run-time checking *)
  | Itry   (* try { *)
  | ICatch (* } catch (exception & e) { cerr << e.what() << endl; } *)

(* target function declare/definite *)
type irfun = {
  ireturn : itype;
  ifname : string;
  iargs : ivar list;
  ibody : irstmt list
}

(* target program *)
type sprogram = {
```

```
  ivars : ivar_dec list;
  ifuns : irfun list
}
```

codegen.ml

```
(* Code Generation
   Input: TAST,
   Output: target code string list *)

open Ast
open Tast
open Printf

exception Not_done of string

(* Translate dtype to C++ types *)
let tpt t = match t with
    Iint | Iint_array -> "int"
  | Idouble | Idouble_array -> "double"
  | Istring | Istring_array -> "string"
  | Ibool -> "bool"
  | Iint_mat -> "int_mat"
  | Idouble_mat -> "double_mat"
  | Istring_mat -> "string_mat"
  | Ivoid -> "void"

let gen_uop op = match op with Neg -> "-"
                        | Not -> "!"

let gen_bop op = match op with
    Plus -> "+" | Minus -> "-" | Times -> "*" | Divide -> "/"
  | Eq -> "==" | Neq -> "!=" | Lt -> "<" | Leq -> "<="
  | Gt -> ">" | Geq -> ">=" | And -> "&&" | Or -> "||"

(* translate expr to string *)
(* @ttbl: (int*sexpr list list) list - temporary variable table *)
(* return: (int*sexpr) list * string - updated ttbl and target code *)
let rec gen_expr exp = match exp with
    IIntval x -> (sprintf " %d " x)
  | IDoubleval x -> (sprintf "%f" x)
  | IStringval x -> (" string(\"" ^ x ^ "\") ")
  | IBoolval x -> if x then " true " else " false "
  | IId x -> (" " ^ x ^ " ")
  | IBinop (e1, b, e2) -> (sprintf "( %s %s %s )" (gen_expr e1) (gen_bop b) (gen_expr e2))
```

```
  | IAssign (e1, e2) -> (sprintf "( %s = %s )" (gen_expr e1) (gen_expr e2))
  | IUnaop (u, e) -> (sprintf "( %s %s )" (gen_uop u) (gen_expr e))
  | ICall (s, el) -> (sprintf "( %s(%s) )" s (gen_arg_list "," el))
  | IArray el->  (sprintf "{%s}" (gen_arg_list "," el) )
  | IIndex (s, e) -> (sprintf " (%s[%s]) " s (gen_expr e))
and gen_arg_list sc el = match el with
    [] -> ""
  | [e] -> gen_expr e
  | e1::e2::tl ->  sprintf "%s %s %s" (gen_expr e1) sc (gen_arg_list sc (e2::tl))

(* translate variable definition list *)
let rec gen_vardecs vars = match vars with
    [] -> []
  | v::tl -> (gen_vardec v) :: (gen_vardecs tl)
and gen_vardec var =
  let t, s, e = var in match t with
    Iint_array | Idouble_array | Istring_array ->
    sprintf "%s %s[] = %s;" (tpt t) s (gen_expr e)
  | t -> sprintf "%s %s = %s;" (tpt t) s (gen_expr e)

let gen_disp es = ("cout << " ^ es ^ " << endl;")

(* translate statement list *)
let rec gen_stmt stmt = match stmt with
    IEmpty -> ";"
  | IVarDec (vt, vn, ve) -> (gen_vardec (vt, vn, ve))
  | IExpr e -> ((gen_expr e) ^ " ;")
  | IReturn e -> (sprintf "return %s ;" (gen_expr e))
  | IIfHead e -> (sprintf "if (%s) {"  (gen_expr e))
  | IElseIf e -> (sprintf "} else if (%s) {"  (gen_expr e))
  | IElse -> (sprintf "} else {")
  | IForHead (e1, e2, e3) -> (sprintf "for (%s %s; %s) {" (gen_stmt e1) (gen_expr e2)
(gen_expr e3))
  | IWhileHead e -> (sprintf "while (%s) {"  (gen_expr e))
  | IBlockEnd -> "}"
  | IDisp e -> (sprintf "cout << %s << endl;" (gen_expr e))
  | IContinue -> "continue;"
  | IBreak -> "break;"
  | ICheck (s, e) -> (sprintf "if (!(%s)) throw invalid_argument(\"%s\");" (gen_expr e) s)
  | Itry -> "try {"
  | ICatch -> "} catch (exception & e) { cerr << e.what() << endl; }"
and gen_stmts stmts = match stmts with
    [] -> []
  | hd::tl -> (gen_stmt hd) :: (gen_stmts tl)
```

```
let rec gen_args sc args = match args with
   [] -> ""
  | [a] -> sprintf "%s %s" (tpt a.ivtype) (a.ivname)
  | a::b::tl -> (sprintf "%s %s%s " (tpt a.ivtype) a.ivname sc) ^ (gen_args sc (b::tl))
let rec gen_fundefs fundefs = match fundefs with
   [] -> []
  | hd::tl -> (if hd.ibody != [] then
              ([sprintf "%s %s(%s) {" (tpt hd.ireturn) hd.ifname (gen_args "," hd.iargs)]
               @(gen_stmts hd.ibody)@ ["}"])
             else ([sprintf "extern %s %s(%s);" (tpt hd.ireturn) hd.ifname (gen_args ","
hd.iargs)])
            )@(gen_fundefs tl)


let compile oc prg =
  let head_lines =
   ["#include \"buckcal_mat.hpp\""; "using namespace std;"]
  in
  let var_lines =
   let vars = prg.ivars in
   gen_vardecs vars
  in
  let func_lines =
   let funs = prg.ifuns in
   gen_fundefs funs
  in
  let all = head_lines @ var_lines @ func_lines in
  (*List.iter print_endline all*)
  List.iter (fun line -> fprintf oc "%s\n" line) all
```

lib.ml

```
open Ast
open Sast

let lib_funs = [
  { sreturn=Int; sfname="rows";
    sargs=[ {vtype=IntMat; vname="mx"}; ];
    slocals=[]; sbody=[SEmpty]
  };
  { sreturn=Int; sfname="rows";
    sargs=[ {vtype=DoubleMat; vname="mx"}; ];
    slocals=[]; sbody=[SEmpty]
  };
  { sreturn=Int; sfname="rows";
```

```
  sargs=[ {vtype=StringMat; vname="mx"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=Int; sfname="cols";
  sargs=[ {vtype=IntMat; vname="mx"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=Int; sfname="cols";
  sargs=[ {vtype=DoubleMat; vname="mx"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=Int; sfname="cols";
  sargs=[ {vtype=StringMat; vname="mx"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=IntMat; sfname="rowcat";
  sargs=[ {vtype=IntMat; vname="mx1"}; {vtype=IntMat; vname="mx2"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=DoubleMat; sfname="rowcat";
  sargs=[ {vtype=DoubleMat; vname="mx1"}; {vtype=DoubleMat; vname="mx2"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=StringMat; sfname="rowcat";
  sargs=[ {vtype=StringMat; vname="mx1"}; {vtype=StringMat; vname="mx2"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=IntMat; sfname="colcat";
  sargs=[ {vtype=IntMat; vname="mx1"}; {vtype=IntMat; vname="mx2"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=DoubleMat; sfname="colcat";
  sargs=[ {vtype=DoubleMat; vname="mx1"}; {vtype=DoubleMat; vname="mx2"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=StringMat; sfname="colcat";
  sargs=[ {vtype=StringMat; vname="mx1"}; {vtype=StringMat; vname="mx2"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=Void; sfname="colunit";
  sargs=[ {vtype=DoubleMat; vname="mx"}; {vtype=StringMat; vname="u"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=Void; sfname="rowname";
  sargs=[ {vtype=IntMat; vname="mx"}; {vtype=StringMat; vname="n"}; ];
```

```
   slocals=[]; sbody=[SEmpty]
  };
  { sreturn=Void; sfname="rowname";
   sargs=[ {vtype=DoubleMat; vname="mx"}; {vtype=StringMat; vname="n"}; ];
   slocals=[]; sbody=[SEmpty]
  };
  { sreturn=Void; sfname="rowname";
   sargs=[ {vtype=StringMat; vname="mx"}; {vtype=StringMat; vname="n"}; ];
   slocals=[]; sbody=[SEmpty]
  };
  { sreturn=Void; sfname="colname";
   sargs=[ {vtype=IntMat; vname="mx"}; {vtype=StringMat; vname="n"}; ];
   slocals=[]; sbody=[SEmpty]
  };
  { sreturn=Void; sfname="colname";
   sargs=[ {vtype=DoubleMat; vname="mx"}; {vtype=StringMat; vname="n"}; ];
   slocals=[]; sbody=[SEmpty]
  };
  { sreturn=Void; sfname="colname";
   sargs=[ {vtype=StringMat; vname="mx"}; {vtype=StringMat; vname="n"}; ];
   slocals=[]; sbody=[SEmpty]
  };
  { sreturn=IntMat; sfname="init_mat";
   sargs=[ {vtype=Int; vname="r"}; {vtype=Int; vname="c"}; {vtype=Int; vname="init"}; ];
   slocals=[]; sbody=[SEmpty]
  };
  { sreturn=DoubleMat; sfname="init_mat";
   sargs=[ {vtype=Int; vname="r"}; {vtype=Int; vname="c"}; {vtype=Double;
vname="init"}; ];
   slocals=[]; sbody=[SEmpty]
  };
  { sreturn=StringMat; sfname="init_mat";
   sargs=[ {vtype=Int; vname="r"}; {vtype=Int; vname="c"}; {vtype=String; vname="init"};
];
   slocals=[]; sbody=[SEmpty]
  };
  { sreturn=String; sfname="string_of_int";
   sargs=[ {vtype=Int; vname="x"}; ];
   slocals=[]; sbody=[SEmpty]
  };
  { sreturn=String; sfname="string_of_double";
   sargs=[ {vtype=Double; vname="x"}; ];
   slocals=[]; sbody=[SEmpty]
  };
  { sreturn=Int; sfname="int_of_string";
```

```
  sargs=[ {vtype=String; vname="x"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=Double; sfname="double_of_string";
  sargs=[ {vtype=String; vname="x"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=IntMat; sfname="mat_int_of_string";
  sargs=[ {vtype=StringMat; vname="x"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=DoubleMat; sfname="mat_double_of_string";
  sargs=[ {vtype=StringMat; vname="x"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=IntMat; sfname="mat_int_of_double";
  sargs=[ {vtype=DoubleMat; vname="x"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=DoubleMat; sfname="mat_double_of_int";
  sargs=[ {vtype=IntMat; vname="x"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=StringMat; sfname="mat_string_of_int";
  sargs=[ {vtype=IntMat; vname="x"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=StringMat; sfname="mat_string_of_double";
  sargs=[ {vtype=DoubleMat; vname="x"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=Int; sfname="strlen";
  sargs=[ {vtype=String; vname="x"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=String; sfname="slice";
  sargs=[ {vtype=String; vname="x"}; {vtype=Int; vname="l"}; {vtype=Int; vname="r"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=IntMat; sfname="getrow";
  sargs=[ {vtype=IntMat; vname="mat"}; {vtype=Int; vname="r"}; ];
  slocals=[]; sbody=[SEmpty]
};
{ sreturn=DoubleMat; sfname="getrow";
  sargs=[ {vtype=DoubleMat; vname="mat"}; {vtype=Int; vname="r"}; ];
```

```
   slocals=[]; sbody=[SEmpty]
  };
  { sreturn=StringMat; sfname="getrow";
   sargs=[ {vtype=StringMat; vname="mat"}; {vtype=Int; vname="r"}; ];
   slocals=[]; sbody=[SEmpty]
  };
  { sreturn=IntMat; sfname="getcol";
   sargs=[ {vtype=IntMat; vname="mat"}; {vtype=Int; vname="c"}; ];
   slocals=[]; sbody=[SEmpty]
  };
  { sreturn=DoubleMat; sfname="getcol";
   sargs=[ {vtype=DoubleMat; vname="mat"}; {vtype=Int; vname="c"}; ];
   slocals=[]; sbody=[SEmpty]
  };
  { sreturn=StringMat; sfname="getcol";
   sargs=[ {vtype=StringMat; vname="mat"}; {vtype=Int; vname="c"}; ];
   slocals=[]; sbody=[SEmpty]
  };
  { sreturn=Void; sfname="setrow";
   sargs=[ {vtype=IntMat; vname="mat"}; {vtype=Int; vname="r"}; {vtype=IntMat;
vname="set"}; ];
   slocals=[]; sbody=[SEmpty]
  };
  { sreturn=Void; sfname="setrow";
   sargs=[ {vtype=DoubleMat; vname="mat"}; {vtype=Int; vname="r"};
{vtype=DoubleMat; vname="set"}; ];
   slocals=[]; sbody=[SEmpty]
  };
  { sreturn=Void; sfname="setrow";
   sargs=[ {vtype=StringMat; vname="mat"}; {vtype=Int; vname="r"}; {vtype=StringMat;
vname="set"}; ];
   slocals=[]; sbody=[SEmpty]
  };
   { sreturn=Void; sfname="setcol";
   sargs=[ {vtype=IntMat; vname="mat"}; {vtype=Int; vname="c"}; {vtype=IntMat;
vname="set"}; ];
   slocals=[]; sbody=[SEmpty]
  };
  { sreturn=Void; sfname="setcol";
   sargs=[ {vtype=DoubleMat; vname="mat"}; {vtype=Int; vname="c"};
{vtype=DoubleMat; vname="set"}; ];
   slocals=[]; sbody=[SEmpty]
  };
  { sreturn=Void; sfname="setcol";
```

```
   sargs=[ {vtype=StringMat; vname="mat"}; {vtype=Int; vname="c"}; {vtype=StringMat;
vname="set"}; ];
   slocals=[]; sbody=[SEmpty]
  };
]
```

main.ml

```
open Printf
open Ast
open Sast


(* helper of error message print *)
let perror head err_msg =
  eprintf "%s: %s\n" head err_msg


(* Paser error reporting functions *)
let loc_err lex_buf =
  let p = lex_buf.Lexing.lex_curr_p in
  let tok = Lexing.lexeme lex_buf in
  let fname = p.Lexing.pos_fname in
  let line = p.Lexing.pos_lnum in
  let cnum = p.Lexing.pos_cnum - p.Lexing.pos_bol + 1
           - String.length tok in
  sprintf "token %s, in %s line %d,%d" tok fname line cnum

(* open input file *)
let get_lex_buf in_file =
   try
     let lexbuf = Lexing.from_channel (open_in in_file) in
     lexbuf.Lexing.lex_curr_p <- {
       lexbuf.Lexing.lex_curr_p with Lexing.pos_fname = in_file
     };
     lexbuf
   with
     Sys_error x -> let msg = sprintf "import %s" x in
                    raise (Ast.Syntax_error msg)

(* Front end - scanner and parser
   return: ast of @main_file *)
let scanner_parser main_file =
  let lex_buf = get_lex_buf main_file in
  try
```

```
    Parser.program Scanner.token lex_buf
  with
    Parsing.Parse_error -> raise (Ast.Syntax_error (loc_err lex_buf))


(* Compile all, main file and imported ones
   @flag : TOP | IMP - main file or imported file
   @main_file : name of file to be compiled
   @ main_oc : out_channel of main file output
*)
let rec compile_all flag main_file main_oc =
  let append_new_sast_funs olds newfile =
    let ofname = (newfile ^ ".cpp") in
    let new_oc = open_out ofname in
    let new_sast = compile_all IMP newfile new_oc in
    let new_sfuns = new_sast.spfuns in
    (olds @ new_sfuns)
  in
  let ast = scanner_parser main_file in
  let newfiles = ast.pimps in
  let extern_funs = List.fold_left append_new_sast_funs [] newfiles in
  let extern_func_table =
    let to_fun_dec ff = { ff with slocals = []; sbody = [] } in
    List.map to_fun_dec extern_funs
  in
  (*let _ = printf "#extern = %d\n" (List.length extern_func_table) in*)
  let sast0 = Scheck.check flag extern_funs ast in
  let sast = { sast0 with spfuns = extern_func_table @ sast0.spfuns } in
  let tast = Translate.translate flag sast in
  let _ = Codegen.compile main_oc tast in
  sast0


(* main function. return 0 on success, 1 on failure *)
let main in_file oc =
  try
    (*let prog = Parser.program Scanner.token lex_buf in*)
    (*let ast = front_end in_file in
    let sast = Scheck.check lib_funs ast*)
    (ignore (compile_all TOP in_file oc); 0)
  with
    Scanner.Scanner_error x -> perror "Scanner error" x; 1
  | Ast.Syntax_error x -> perror "Parser error" x; 1
  | Sast.Bad_type x -> perror "Sast error" x; 1
  | Tast.Not_now x -> perror "Translate error" x; 1
```

```
  | Codegen.Not_done x -> perror "Codegen error" x; 1


(* Shell interface *)
let () =
  let argc = Array.length Sys.argv in
  let exit_code =
    if argc >= 2 then
      let oc =
        let ofile =
          if argc >= 3 then Sys.argv.(2) else "buckcal_out.cpp"
        in
        open_out ofile
      in
      main Sys.argv.(1) oc
    else
      (eprintf "Usage: main.bin <input file>\n"; 1)
  in
  exit exit_code
```

buckcal_lib.hpp

```
#ifndef _BUCKCAL_HPP_
#define _BUCKCAL_HPP_
#include <iostream>
#include <vector>
#include <fstream>
#include <string>
#include <stdexcept>
#include <sstream>

class int_mat;
class double_mat;
class string_mat;

class double_mat {
public:
        int rows;
        int cols;
        std::vector<double> m;
        std::vector<std::string> rownames;
        std::vector<std::string> colnames;
        /* constructors */
        double_mat(double *, int, int);
        double_mat(const double_mat &);
```

```
        double_mat(const int_mat &);
        /* index */
        double & operator [] (int);
        /* overload = and << */
        double_mat & operator = (const double_mat &);
        double_mat & operator = (const int_mat &);
        double_mat operator + (const double_mat &);
        double_mat operator + (const double &);
        double_mat operator - (const double_mat &);
        double_mat operator - (const double &);
        double_mat operator * (const double_mat &);
        double_mat operator * (const double &);
        double_mat operator / (const double_mat &);
        double_mat operator / (const double &);
        double_mat operator + (const int_mat &);
        double_mat operator + (const int &);
        double_mat operator - (const int_mat &);
        double_mat operator - (const int &);
        double_mat operator * (const int_mat &);
        double_mat operator * (const int &);
        double_mat operator / (const int_mat &);
        double_mat operator / (const int &);
};

class int_mat {
public:
        int rows;
        int cols;
        std::vector<int> m;
        std::vector<std::string> rownames;
        std::vector<std::string> colnames;
        /* constructors */
        int_mat(int *, int, int);
        int_mat(const int_mat &);
        int_mat(const double_mat &);
        /* index */
        int & operator [] (int);
        /* overload = and << */
        int_mat & operator = (const int_mat &);
        int_mat & operator = (const double_mat &);
        int_mat operator + (const int_mat &);
        int_mat operator + (const int &);
        int_mat operator - (const int_mat &);
        int_mat operator - (const int &);
        int_mat operator * (const int_mat &);
```

```
        int_mat operator * (const int &);
        int_mat operator / (const int_mat &);
        int_mat operator / (const int &);
        double_mat operator + (const double_mat &);
        double_mat operator + (const double &);
        double_mat operator - (const double_mat &);
        double_mat operator - (const double &);
        double_mat operator * (const double_mat &);
        double_mat operator * (const double &);
        double_mat operator / (const double_mat &);
        double_mat operator / (const double &);
};

class string_mat {
public:
        int rows;
        int cols;
        std::vector<std::string> m;
        std::vector<std::string> rownames;
        std::vector<std::string> colnames;
        /* constructors */
        string_mat(std::string *, int, int);
        string_mat(const string_mat &);
        /* index */
        std::string & operator [] (int);
        /* overload = and << */
        string_mat & operator = (const string_mat &);
};

/* for cout */
std::ostream & operator << (std::ostream &sys, const int_mat &in);
std::ostream & operator << (std::ostream &sys, const double_mat &in);
std::ostream & operator << (std::ostream &sys, const string_mat &in);

/* get rows and columns */
int rows(int_mat mx);
int rows(double_mat mx);
int rows(string_mat mx);

int cols(int_mat mx);
int cols(double_mat mx);
int cols(string_mat mx);

/* data conversion */
std::string string_of_int(int x);
```

```
std::string string_of_double(double x);
int int_of_string(std::string x);
double double_of_string(std::string x);
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);

/* row and column concatenation */
int_mat rowcat(int_mat mx1, int_mat mx2);
double_mat rowcat(double_mat mx1, 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 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);

/* row and column names */
void rowname(int_mat &mx, string_mat n);
void rowname(double_mat &mx, string_mat n);
void rowname(string_mat &mx, string_mat n);

void colname(int_mat &mx, string_mat n);
void colname(double_mat &mx, string_mat n);
void colname(string_mat &mx, string_mat n);

/* string operations */
int strlen(std::string x);
std::string slice(std::string x, int l, int r);

/* get or set row/col */
int_mat getrow(int_mat mat, int r);
double_mat getrow(double_mat mat, int r);
string_mat getrow(string_mat mat, int r);

void setrow(int_mat &mat, int r, int_mat set);
void setrow(double_mat &mat, int r, double_mat set);
void setrow(string_mat &mat, int r, string_mat set);
void setrow(int_mat &mat, int r, double_mat set);
```

```
void setrow(double_mat &mat, int r, int_mat set);

int_mat getcol(int_mat mat, int c);
double_mat getcol(double_mat mat, int c);
string_mat getcol(string_mat mat, int c);

void setcol(int_mat &mat, int c, int_mat set);
void setcol(double_mat &mat, int c, double_mat set);
void setcol(string_mat &mat, int c, string_mat set);
void setcol(int_mat &mat, int c, double_mat set);
void setcol(double_mat &mat, int c, int_mat set);

/* init matrixes */
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, std::string init);
#endif
```

buckcal_mat.cpp

```
#include "buckcal_mat.hpp"

using namespace std;

void init_names(vector<string> &rownames, vector<string> &colnames, int r, int c) {
	for (int i = 0; i < r; i++) {
		ostringstream ss;
		ss << "r" << (i + 1);
		rownames.push_back(ss.str());
	}
	for (int i = 0; i < c; i++) {
		ostringstream ss;
		ss << "c" << (i + 1);
		colnames.push_back(ss.str());
	}
}

int_mat::int_mat(int *array, int r, int c) {
	rows = r;
	cols = c;
	for (int i = 0; i < r * c; i++)
		m.push_back(array[i]);
	init_names(rownames, colnames, rows, cols);
}
```

```
int_mat::int_mat(const int_mat &in) {
	if (this == &in)
		return;
	rows = in.rows;
	cols = in.cols;
	m = in.m;
	rownames = in.rownames;
	colnames = in.colnames;
}

int_mat::int_mat(const double_mat &in) {
	rows = in.rows;
	cols = in.cols;
	m.resize(rows*cols);
	for (int i = 0; i < rows * cols; i++)
		m[i] = in.m[i];
	rownames = in.rownames;
	colnames = in.colnames;
}

int & int_mat::operator [] (int i) {
	return m.at(i);
}

int_mat & int_mat::operator = (const int_mat &in) {
	if (this == &in)
		return *this;
	rows = in.rows;
	cols = in.cols;
	m = in.m;
	rownames = in.rownames;
	colnames = in.colnames;
	return *this;
}

int_mat & int_mat::operator = (const double_mat &in) {
	return operator = ((int_mat) in);
}

int_mat int_mat::operator + (const int_mat &in) {
	int_mat result = *this;
	if (rows != in.rows || cols != in.cols)
		throw std::invalid_argument("operator '+': matrixes dimensions do not
match");
	for (int i = 0; i < rows*cols; i++)
```

```
                result.m[i] += in.m[i];
        return result;
}

int_mat int_mat::operator + (const int &in) {
        int_mat result = *this;
        for (int i = 0; i < rows*cols; i++)
                result.m[i] += in;
        return result;
}

int_mat int_mat::operator - (const int_mat &in) {
        int_mat result = *this;
        if (rows != in.rows || cols != in.cols)
                throw std::invalid_argument("operator '-': matrixes dimensions do not
match");
        for (int i = 0; i < rows*cols; i++)
                result.m[i] -= in.m[i];
        return result;
}

int_mat int_mat::operator - (const int &in) {
        int_mat result = *this;
        for (int i = 0; i < rows*cols; i++)
                result.m[i] -= in;
        return result;
}

int_mat int_mat::operator * (const int_mat &in) {
        int_mat result = *this;
        if (rows != in.rows || cols != in.cols)
                throw std::invalid_argument("operator '*': matrixes dimensions do not
match");
        for (int i = 0; i < rows*cols; i++)
                result.m[i] *= in.m[i];
        return result;
}

int_mat int_mat::operator * (const int &in) {
        int_mat result = *this;
        for (int i = 0; i < rows*cols; i++)
                result.m[i] *= in;
        return result;
}
```

```
int_mat int_mat::operator / (const int_mat &in) {
	int_mat result = *this;
	if (rows != in.rows || cols != in.cols)
		throw std::invalid_argument("operator '/': matrixes dimensions do not
match");
	for (int i = 0; i < rows*cols; i++)
		result.m[i] /= in.m[i];
	return result;
}

int_mat int_mat::operator / (const int &in) {
	int_mat result = *this;
	for (int i = 0; i < rows*cols; i++)
		result.m[i] /= in;
	return result;
}

double_mat int_mat::operator + (const double_mat &in) {
	double_mat result = (double_mat) *this;
	return result.operator + (in);
}

double_mat int_mat::operator + (const double &in) {
	double_mat result = (double_mat) *this;
	return result.operator + (in);
}

double_mat int_mat::operator - (const double_mat &in) {
	double_mat result = (double_mat) *this;
	return result.operator - (in);
}

double_mat int_mat::operator - (const double &in) {
	double_mat result = (double_mat) *this;
	return result.operator - (in);
}

double_mat int_mat::operator * (const double_mat &in) {
	double_mat result = (double_mat) *this;
	return result.operator * (in);
}

double_mat int_mat::operator * (const double &in) {
	double_mat result = (double_mat) *this;
	return result.operator * (in);
```

```
}

double_mat int_mat::operator / (const double_mat &in) {
	double_mat result = (double_mat) *this;
	return result.operator / (in);
}

double_mat int_mat::operator / (const double &in) {
	double_mat result = (double_mat) *this;
	return result.operator / (in);
}

ostream & operator << (ostream &sys, const int_mat &in) {
	sys << "\t";
	for (int i = 0; i < in.cols; i++)
		sys << in.colnames[i] << "\t";
	sys << endl;
	for (int i = 0; i < in.rows; i++) {
		sys << in.rownames[i] << "\t[";
		for (int j = 0; j < in.cols; j++) {
			sys << in.m[i * in.cols + j];
			if (j < in.cols - 1)
				sys << ",\t";
		}
		sys << "]" <<endl;
	}
	return sys;
}

double_mat::double_mat(double *array, int r, int c) {
	rows = r;
	cols = c;
	for (int i = 0; i < r * c; i++)
		m.push_back(array[i]);
	init_names(rownames, colnames, rows, cols);
}

double_mat::double_mat(const double_mat &in) {
	if (this == &in)
		return;
	rows = in.rows;
	cols = in.cols;
	m = in.m;
	rownames = in.rownames;
	colnames = in.colnames;
```

```
}
double_mat::double_mat(const int_mat &in) {
	rows = in.rows;
	cols = in.cols;
	m.resize(rows*cols);
	for (int i = 0; i < rows * cols; i++)
		m[i] = in.m[i];
	rownames = in.rownames;
	colnames = in.colnames;
}
double & double_mat::operator [] (int i) {
	return m.at(i);
}
double_mat & double_mat::operator = (const double_mat &in) {
	if (this == &in)
		return *this;
	rows = in.rows;
	cols = in.cols;
	/* copy array */
	m = in.m;
	rownames = in.rownames;
	colnames = in.colnames;
	return *this;
}
double_mat & double_mat::operator = (const int_mat &in) {
	return operator = ((double_mat) in);
}
double_mat double_mat::operator + (const double_mat &in) {
	double_mat result = *this;
	if (rows != in.rows || cols != in.cols)
		throw std::invalid_argument("operator '+': matrixes dimensions do not
match");
	for (int i = 0; i < rows*cols; i++)
		result.m[i] += in.m[i];
	return result;
}

double_mat double_mat::operator + (const double &in) {
	double_mat result = *this;
	for (int i = 0; i < rows*cols; i++)
		result.m[i] += in;
	return result;
}

double_mat double_mat::operator - (const double_mat &in) {
```

```
	double_mat result = *this;
	if (rows != in.rows || cols != in.cols)
		throw std::invalid_argument("operator '-': matrixes dimensions do not
match");
	for (int i = 0; i < rows*cols; i++)
		result.m[i] -= in.m[i];
	return result;
}

double_mat double_mat::operator - (const double &in) {
	double_mat result = *this;
	for (int i = 0; i < rows*cols; i++)
		result.m[i] -= in;
	return result;
}

double_mat double_mat::operator * (const double_mat &in) {
	double_mat result = *this;
	if (rows != in.rows || cols != in.cols)
		throw std::invalid_argument("operator '*': matrixes dimensions do not
match");
	for (int i = 0; i < rows*cols; i++)
		result.m[i] *= in.m[i];
	return result;
}

double_mat double_mat::operator * (const double &in) {
	double_mat result = *this;
	for (int i = 0; i < rows*cols; i++)
		result.m[i] *= in;
	return result;
}

double_mat double_mat::operator / (const double_mat &in) {
	double_mat result = *this;
	if (rows != in.rows || cols != in.cols)
		throw std::invalid_argument("operator '/': matrixes dimensions do not
match");
	for (int i = 0; i < rows*cols; i++)
		result.m[i] /= in.m[i];
	return result;
}

double_mat double_mat::operator / (const double &in) {
	double_mat result = *this;
```

```
	for (int i = 0; i < rows*cols; i++)
		result.m[i] /= in;
	return result;
}

double_mat double_mat::operator + (const int_mat &in) {
	return operator + ((double_mat) in);
}

double_mat double_mat::operator + (const int &in) {
	return operator + ((double) in);
}

double_mat double_mat::operator - (const int_mat &in) {
	return operator - ((double_mat) in);
}

double_mat double_mat::operator - (const int &in) {
	return operator - ((double) in);
}

double_mat double_mat::operator * (const int_mat &in) {
	return operator * ((double_mat) in);
}

double_mat double_mat::operator * (const int &in) {
	return operator * ((double) in);
}

double_mat double_mat::operator / (const int_mat &in) {
	return operator / ((double_mat) in);
}

double_mat double_mat::operator / (const int &in) {
	return operator / ((double) in);
}

ostream & operator << (ostream &sys, const double_mat &in) {
	sys << "\t";
	for (int i = 0; i < in.cols; i++)
		sys << in.colnames[i] << "\t";
	sys << endl;
	for (int i = 0; i < in.rows; i++) {
		sys << in.rownames[i] << "\t[";
		for (int j = 0; j < in.cols; j++) {
```

```
                        sys << in.m[i * in.cols + j];
                        if (j < in.cols - 1)
                                sys << ",\t";
                }
                sys << "]" <<endl;
        }
        return sys;
}

string_mat::string_mat(string *array, int r, int c) {
        rows = r;
        cols = c;
        for (int i = 0; i < r * c; i++)
                m.push_back(array[i]);
        init_names(rownames, colnames, rows, cols);
}

string_mat::string_mat(const string_mat &in) {
        if (this == &in)
                return;
        rows = in.rows;
        cols = in.cols;
        m = in.m;
        rownames = in.rownames;
        colnames = in.colnames;
}
string & string_mat::operator [] (int i) {
        return m.at(i);
}
string_mat & string_mat::operator = (const string_mat &in) {
        if (this == &in)
                return *this;
        rows = in.rows;
        cols = in.cols;
        m.resize(rows*cols);
        for (int i = 0; i < rows * cols; i++)
                m[i] = in.m[i];
        rownames = in.rownames;
        colnames = in.colnames;
        return *this;
}

ostream & operator << (ostream &sys, const string_mat &in) {
        sys << "\t";
        for (int i = 0; i < in.cols; i++)
```

```
                sys << in.colnames[i] << "\t";
        sys << endl;
        for (int i = 0; i < in.rows; i++) {
                sys << in.rownames[i] << "\t[";
                for (int j = 0; j < in.cols; j++) {
                        sys << in.m[i * in.cols + j];
                        if (j < in.cols - 1)
                                sys << ",\t";
                }
                sys << "]" <<endl;
        }
        return sys;
}
```

buckcal_core.cpp

```
#include "buckcal_mat.hpp"
#include <cstdlib>

using namespace std;

/* get number of rows and columns */
int rows(int_mat mx) {
        return mx.rows;
}

int rows(double_mat mx) {
        return mx.rows;
}

int rows(string_mat mx) {
        return mx.rows;
}

int cols(int_mat mx) {
        return mx.cols;
}

int cols(double_mat mx) {
        return mx.cols;
}

int cols(string_mat mx) {
        return mx.cols;
}
```

```
/* data conversion */
string string_of_int(int x) {
	ostringstream ss;
	ss << x;
	return ss.str();
}

string string_of_double(double x) {
	ostringstream ss;
	ss << x;
	return ss.str();
}

int int_of_string(string x) {
	return atoi(x.c_str());
}

double double_of_string(string x) {
	return atof(x.c_str());
}

int_mat mat_int_of_string(string_mat x) {
	int *array = new int[x.rows * x.cols];
	for (int i = 0; i < x.rows * x.cols; i++)
		array[i] = atoi(x.m[i].c_str());
	int_mat mat(array, x.rows, x.cols);
	mat.rownames = x.rownames;
	mat.colnames = x.colnames;
	delete[] array;
	return mat;
}

double_mat mat_double_of_string(string_mat x) {
	double *array = new double[x.rows * x.cols];
	for (int i = 0; i < x.rows * x.cols; i++)
		array[i] = atof(x.m[i].c_str());
	double_mat mat(array, x.rows, x.cols);
	mat.rownames = x.rownames;
	mat.colnames = x.colnames;
	delete[] array;
	return mat;
}

string_mat mat_string_of_int(int_mat x) {
```

```
	string *array = new string[x.rows * x.cols];
	for (int i = 0; i < x.rows * x.cols; i++) {
		ostringstream ss;
		ss << x.m[i];
		array[i] = ss.str();
	}
	string_mat mat(array, x.rows, x.cols);
	mat.rownames = x.rownames;
	mat.colnames = x.colnames;
	delete[] array;
	return mat;
}

string_mat mat_string_of_double(double_mat x) {
	string *array = new string[x.rows * x.cols];
	for (int i = 0; i < x.rows * x.cols; i++) {
		ostringstream ss;
		ss << x.m[i];
		array[i] = ss.str();
	}
	string_mat mat(array, x.rows, x.cols);
	mat.rownames = x.rownames;
	mat.colnames = x.colnames;
	delete[] array;
	return mat;
}

int_mat mat_int_of_double(double_mat x) {
	int *array = new int[x.rows * x.cols];
	for (int i = 0; i < x.rows * x.cols; i++)
		array[i] = (int) x.m[i];
	int_mat mat(array, x.rows, x.cols);
	mat.rownames = x.rownames;
	mat.colnames = x.colnames;
	delete[] array;
	return mat;
}

double_mat mat_double_of_int(int_mat x)  {
	double *array = new double[x.rows * x.cols];
	for (int i = 0; i < x.rows * x.cols; i++)
		array[i] = (double) x.m[i];
	double_mat mat(array, x.rows, x.cols);
	mat.rownames = x.rownames;
	mat.colnames = x.colnames;
```

```
	delete[] array;
	return mat;
}

/* row and column concatenation */
int_mat rowcat(int_mat mx1, int_mat mx2) {
	if (mx1.rows == 0 || mx1.cols == 0)
		return mx2;
	if (mx2.rows == 0 || mx2.cols == 0)
		return mx1;
	if (mx1.cols != mx2.cols)
		throw std::invalid_argument("rowcat: matrix does not have the same
number of columns");
	int *array = new int[mx1.rows * mx1.cols + mx2.rows * mx2.cols];
	for (int i = 0; i < mx1.rows * mx1.cols; i++)
		array[i] = mx1.m[i];
	for (int i = 0; i < mx2.rows * mx2.cols; i++)
		array[i + mx1.rows * mx1.cols] = mx2.m[i];
	int_mat mat(array, mx1.rows + mx2.rows, mx1.cols);
	delete[] array;
	mat.colnames = mx1.colnames;
	mat.rownames = mx1.rownames;
	mat.rownames.insert(mat.rownames.end(), mx2.rownames.begin(),
mx2.rownames.end());
	return mat;
}

double_mat rowcat(double_mat mx1, double_mat mx2) {
	if (mx1.rows == 0 || mx1.cols == 0)
		return mx2;
	if (mx2.rows == 0 || mx2.cols == 0)
		return mx1;
	if (mx1.cols != mx2.cols)
		throw std::invalid_argument("rowcat: matrix does not have the same
number of columns");
	double *array = new double[mx1.rows * mx1.cols + mx2.rows * mx2.cols];
	for (int i = 0; i < mx1.rows * mx1.cols; i++)
		array[i] = mx1.m[i];
	for (int i = 0; i < mx2.rows * mx2.cols; i++)
		array[i + mx1.rows * mx1.cols] = mx2.m[i];
	double_mat mat(array, mx1.rows + mx2.rows, mx1.cols);
	delete[] array;
	mat.colnames = mx1.colnames;
	mat.rownames = mx1.rownames;
```

```
	mat.rownames.insert(mat.rownames.end(), mx2.rownames.begin(),
mx2.rownames.end());
	return mat;
}

int_mat rowcat(int_mat mx1, double_mat mx2) {
	return rowcat(mx1, (int_mat) mx2);
}

double_mat rowcat(double_mat mx1, int_mat mx2) {
	return rowcat(mx1, (double_mat) mx2);
}

string_mat rowcat(string_mat mx1, string_mat mx2) {
	if (mx1.rows == 0 || mx1.cols == 0)
		return mx2;
	if (mx2.rows == 0 || mx2.cols == 0)
		return mx1;
	if (mx1.cols != mx2.cols)
		throw std::invalid_argument("rowcat: matrix does not have the same
number of columns");
	string *array = new string[mx1.rows * mx1.cols + mx2.rows * mx2.cols];
	for (int i = 0; i < mx1.rows * mx1.cols; i++)
		array[i] = mx1.m[i];
	for (int i = 0; i < mx2.rows * mx2.cols; i++)
		array[i + mx1.rows * mx1.cols] = mx2.m[i];
	string_mat mat(array, mx1.rows + mx2.rows, mx1.cols);
	delete[] array;
	mat.colnames = mx1.colnames;
	mat.rownames = mx1.rownames;
	mat.rownames.insert(mat.rownames.end(), mx2.rownames.begin(),
mx2.rownames.end());
	return mat;
}

int_mat colcat(int_mat mx1, int_mat mx2) {
	if (mx1.rows == 0 || mx1.cols == 0)
		return mx2;
	if (mx2.rows == 0 || mx2.cols == 0)
		return mx1;
	if (mx1.rows != mx2.rows)
		throw std::invalid_argument("colcat: matrix does not have the same
number of rows");
	int *array = new int[mx1.rows * mx1.cols + mx2.rows * mx2.cols];
	for (int i = 0; i < mx1.rows; i++) {
```

```
                for (int j = 0; j < mx1.cols; j++)
                        array[i * (mx1.cols + mx2.cols) + j] = mx1.m[i * mx1.cols + j];
                for (int j = 0; j < mx2.cols; j++)
                        array[i * (mx1.cols + mx2.cols) + mx1.cols + j] = mx2.m[i * mx2.cols
+ j];
        }
        int_mat mat(array, mx1.rows, mx1.cols + mx2.cols);
        delete[] array;
        mat.rownames = mx1.rownames;
        mat.colnames = mx1.colnames;
        mat.colnames.insert(mat.colnames.end(), mx2.colnames.begin(),
mx2.colnames.end());
        return mat;
}

double_mat colcat(double_mat mx1, double_mat mx2) {
        if (mx1.rows == 0 || mx1.cols == 0)
                return mx2;
        if (mx2.rows == 0 || mx2.cols == 0)
                return mx1;
        if (mx1.rows != mx2.rows)
                throw std::invalid_argument("colcat: matrix does not have the same
number of rows");
        double *array = new double[mx1.rows * mx1.cols + mx2.rows * mx2.cols];
        for (int i = 0; i < mx1.rows; i++) {
                for (int j = 0; j < mx1.cols; j++)
                        array[i * (mx1.cols + mx2.cols) + j] = mx1.m[i * mx1.cols + j];
                for (int j = 0; j < mx2.cols; j++)
                        array[i * (mx1.cols + mx2.cols) + mx1.cols + j] = mx2.m[i * mx2.cols
+ j];
        }
        double_mat mat(array, mx1.rows, mx1.cols + mx2.cols);
        delete[] array;
        mat.rownames = mx1.rownames;
        mat.colnames = mx1.colnames;
        mat.colnames.insert(mat.colnames.end(), mx2.colnames.begin(),
mx2.colnames.end());
        return mat;
}

int_mat colcat(int_mat mx1, double_mat mx2) {
        return colcat(mx1, (int_mat) mx2);
}

double_mat colcat(double_mat mx1, int_mat mx2) {
```

```
	return colcat(mx1, (double_mat) mx2);
}

string_mat colcat(string_mat mx1, string_mat mx2) {
	if (mx1.rows == 0 || mx1.cols == 0)
		return mx2;
	if (mx2.rows == 0 || mx2.cols == 0)
		return mx1;
	if (mx1.rows != mx2.rows)
		throw std::invalid_argument("colcat: matrix does not have the same
number of rows");
	string *array = new string[mx1.rows * mx1.cols + mx2.rows * mx2.cols];
	for (int i = 0; i < mx1.rows; i++) {
		for (int j = 0; j < mx1.cols; j++)
			array[i * (mx1.cols + mx2.cols) + j] = mx1.m[i * mx1.cols + j];
		for (int j = 0; j < mx2.cols; j++)
			array[i * (mx1.cols + mx2.cols) + mx1.cols + j] = mx2.m[i * mx2.cols
+ j];
	}
	string_mat mat(array, mx1.rows, mx1.cols + mx2.cols);
	delete[] array;
	mat.rownames = mx1.rownames;
	mat.colnames = mx1.colnames;
	mat.colnames.insert(mat.colnames.end(), mx2.colnames.begin(),
mx2.colnames.end());
	return mat;
}

/* row and column names */
void rowname(int_mat &mx, string_mat n) {
	if (mx.rows != n.cols)
		throw std::invalid_argument("rowname: name matrix does not have
enough entries");
	if (n.rows > 1)
		throw std::invalid_argument("rowname: name matrix should have only
one row");
	mx.rownames = n.m;
}

void rowname(double_mat &mx, string_mat n) {
	if (mx.rows != n.cols)
		throw std::invalid_argument("rowname: name matrix does not have
enough entries");
	if (n.rows > 1)
```

```
		throw std::invalid_argument("rowname: name matrix should have only
one row");
	mx.rownames = n.m;
}

void rowname(string_mat &mx, string_mat n) {
	if (mx.rows != n.cols)
		throw std::invalid_argument("rowname: name matrix does not have
enough entries");
	if (n.rows > 1)
		throw std::invalid_argument("rowname: name matrix should have only
one row");
	mx.rownames = n.m;
}

void colname(int_mat &mx, string_mat n) {
	if (mx.cols != n.cols)
		throw std::invalid_argument("rowname: name matrix does not have
matching entries");
	if (n.rows > 1)
		throw std::invalid_argument("rowname: name matrix should have only
one row");
	mx.colnames = n.m;
}

void colname(double_mat &mx, string_mat n) {
	if (mx.cols != n.cols)
		throw std::invalid_argument("rowname: name matrix does not have
matching entries");
	if (n.rows > 1)
		throw std::invalid_argument("rowname: name matrix should have only
one row");
	mx.colnames = n.m;
}

void colname(string_mat &mx, string_mat n) {
	if (mx.cols != n.cols)
		throw std::invalid_argument("rowname: name matrix does not have
matching entries");
	if (n.rows > 1)
		throw std::invalid_argument("rowname: name matrix should have only
one row");
	mx.colnames = n.m;
}
```

```
/* string operations */
int strlen(string x) {
	return x.length();
}

string slice(string x, int l, int r) {
	if (r <= l || l <= 0 || r <= 0 || l > (int) x.length() || r > (int) x.length() + 1)
		throw std::invalid_argument("slice: invalid range");
	return x.substr(l + 1, r + 1);
}

/* get or set row/col */
int_mat getrow(int_mat mat, int r) {
	if (r > mat.rows || r < 1)
		throw std::invalid_argument("getrow: argument not in row range");
	int *array = new int[mat.cols];
	for (int i = 0; i < mat.cols; i++)
		array[i] = mat.m[(r - 1) * mat.cols + i];
	int_mat new_mat = int_mat(array, 1, mat.cols);
	new_mat.rownames[0] = mat.rownames[r - 1];
	new_mat.colnames = mat.colnames;
	delete[] array;
	return new_mat;
}

double_mat getrow(double_mat mat, int r) {
	if (r > mat.rows || r < 1)
		throw std::invalid_argument("getrow: argument row not in row range");
	double *array = new double[mat.cols];
	for (int i = 0; i < mat.cols; i++)
		array[i] = mat.m[(r - 1) * mat.cols + i];
	double_mat new_mat = double_mat(array, 1, mat.cols);
	new_mat.rownames[0] = mat.rownames[r - 1];
	new_mat.colnames = mat.colnames;
	delete[] array;
	return new_mat;
}

string_mat getrow(string_mat mat, int r) {
	if (r > mat.rows || r < 1)
		throw std::invalid_argument("getrow: argument row not in row range");
	string *array = new string[mat.cols];
	for (int i = 0; i < mat.cols; i++)
		array[i] = mat.m[(r - 1) * mat.cols + i];
	string_mat new_mat = string_mat(array, 1, mat.cols);
```

```
	new_mat.rownames[0] = mat.rownames[r - 1];
	new_mat.colnames = mat.colnames;
	delete[] array;
	return new_mat;
}

void setrow(int_mat &mat, int r, int_mat set) {
	if (r > mat.rows || r < 1)
		throw std::invalid_argument("setrow: argument row not in row range");

	if (mat.cols != set.cols)
		throw std::invalid_argument("setrow: matrix to be set does not have
matching column number");
	if (set.rows > 1)
		throw std::invalid_argument("setrow: input matrix has more than one
row");
	for (int i = 0; i < mat.cols; i++)
		mat.m[(r - 1) * mat.cols + i] = set.m[i];
	mat.rownames[r - 1] = set.rownames[0];
}

void setrow(double_mat &mat, int r, double_mat set) {
	if (r > mat.rows || r < 1)
		throw std::invalid_argument("setrow: argument row not in row range");

	if (mat.cols != set.cols)
		throw std::invalid_argument("setrow: matrix to be set does not have
matching column number");
	if (set.rows > 1)
		throw std::invalid_argument("setrow: input matrix has more than one
row");
	for (int i = 0; i < mat.cols; i++)
		mat.m[(r - 1) * mat.cols + i] = set.m[i];
	mat.rownames[r - 1] = set.rownames[0];
}

void setrow(string_mat &mat, int r, string_mat set) {
	if (r > mat.rows || r < 1)
		throw std::invalid_argument("setrow: argument row not in row range");

	if (mat.cols != set.cols)
		throw std::invalid_argument("setrow: matrix to be set does not have
matching column number");
	if (set.rows > 1)
```

```
                throw std::invalid_argument("setrow: input matrix has more than one
row");
        for (int i = 0; i < mat.cols; i++)
                mat.m[(r - 1) * mat.cols + i] = set.m[i];
        mat.rownames[r - 1] = set.rownames[0];
}

void setrow(int_mat &mat, int r, double_mat set) {
        setrow(mat, r, (int_mat) set);
}

void setrow(double_mat &mat, int r, int_mat set) {
        setrow(mat, r, (double_mat) set);
}

int_mat getcol(int_mat mat, int c) {
        if (c > mat.cols || c < 1)
                throw std::invalid_argument("getcol: argument not in col range");
        int *array = new int[mat.rows];
        for (int i = 0; i < mat.rows; i++)
                array[i] = mat.m[i * mat.cols + (c - 1)];
        int_mat new_mat = int_mat(array, mat.rows, 1);
        new_mat.colnames[0] = mat.colnames[c - 1];
        new_mat.rownames = mat.rownames;
        delete[] array;
        return new_mat;
}

double_mat getcol(double_mat mat, int c) {
        if (c > mat.cols || c < 1)
                throw std::invalid_argument("getcol: argument not in col range");
        double *array = new double[mat.rows];
        for (int i = 0; i < mat.rows; i++)
                array[i] = mat.m[i * mat.cols + (c - 1)];
        double_mat new_mat = double_mat(array, mat.rows, 1);
        new_mat.colnames[0] = mat.colnames[c - 1];
        new_mat.rownames = mat.rownames;
        delete[] array;
        return new_mat;
}

string_mat getcol(string_mat mat, int c) {
        if (c > mat.cols || c < 1)
                throw std::invalid_argument("getcol: argument not in col range");
        string *array = new string[mat.rows];
```

```
	for (int i = 0; i < mat.rows; i++)
		array[i] = mat.m[i * mat.cols + (c - 1)];
	string_mat new_mat = string_mat(array, mat.rows, 1);
	new_mat.colnames[0] = mat.colnames[c - 1];
	new_mat.rownames = mat.rownames;
	delete[] array;
	return new_mat;
}

void setcol(int_mat &mat, int c, int_mat set) {
	if (c > mat.cols || c < 1)
		throw std::invalid_argument("setcol: argument row not in column range");

	if (mat.rows != set.rows)
		throw std::invalid_argument("setcol: matrix to be set does not have
matching row number");
	if (set.cols > 1)
		throw std::invalid_argument("setcol: input matrix has more than one
column");
	for (int i = 0; i < mat.rows; i++)
		mat.m[i * mat.cols + (c - 1)] = set.m[i];
	mat.colnames[c - 1] = set.colnames[0];
}

void setcol(double_mat &mat, int c, double_mat set) {
	if (c > mat.cols || c < 1)
		throw std::invalid_argument("setcol: argument row not in column range");

	if (mat.rows != set.rows)
		throw std::invalid_argument("setcol: matrix to be set does not have
matching row number");
	if (set.cols > 1)
		throw std::invalid_argument("setcol: input matrix has more than one
column");
	for (int i = 0; i < mat.rows; i++)
		mat.m[i * mat.cols + (c - 1)] = set.m[i];
	mat.colnames[c - 1] = set.colnames[0];
}

void setcol(string_mat &mat, int c, string_mat set) {
	if (c > mat.cols || c < 1)
		throw std::invalid_argument("setcol: argument row not in column range");

	if (mat.rows != set.rows)
```

```
		throw std::invalid_argument("setcol: matrix to be set does not have
matching row number");
	if (set.cols > 1)
		throw std::invalid_argument("setcol: input matrix has more than one
column");
	for (int i = 0; i < mat.rows; i++)
		mat.m[i * mat.cols + (c - 1)] = set.m[i];
	mat.colnames[c - 1] = set.colnames[0];
}

void setcol(int_mat &mat, int c, double_mat set) {
	setcol(mat, c, (int_mat) set);
}

void setcol(double_mat &mat, int c, int_mat set) {
	setcol(mat, c, (double_mat) set);
}

/* init matrixes */
int_mat init_mat(int r, int c, int init) {
	if (r < 0 || c < 0)
		throw std::invalid_argument("init_mat: invalid row and column");
	int *array = new int[r * c];
	for (int i = 0; i < r * c; i++)
		array[i] = init;
	int_mat mat = int_mat(array, r, c);
	delete[] array;
	return mat;
}

double_mat init_mat(int r, int c, double init) {
	if (r < 0 || c < 0)
		throw std::invalid_argument("init_mat: invalid row and column");
	double *array = new double[r * c];
	for (int i = 0; i < r * c; i++)
		array[i] = init;
	double_mat mat = double_mat(array, r, c);
	delete[] array;
	return mat;
}

string_mat init_mat(int r, int c, string init) {
	if (r < 0 || c < 0)
		throw std::invalid_argument("init_mat: invalid row and column");
	string *array = new string[r * c];
```

```
        for (int i = 0; i < r * c; i++)
                array[i] = init;
        string_mat mat = string_mat(array, r, c);
        delete[] array;
        return mat;
}
```

buckcal_lib.bc

```
# function: find the absolute value of a given double
def double abs(double x) do
        if x < 0 then
                return (-1 * x);
        fi
        return x;
fed

#function: Return a column vector {x; x+1; ... ; y}. If  y < x, return {}

def int mat range(int x, int y) do
        int mat v;
        int i : x;
        int mat tmp;
        for (y >= i) do
                tmp: {i};
                colname(tmp, {'c' + string_of_int(i)});
                v : colcat(v, tmp);
                i: i + 1;
        rof
        return v;
fed

#function: Return the number of rows in matrix mx;
def int mat range_col(int x, int y) do
        int mat v;
        int i : x;
        int mat tmp;
        for (y >= i) do
                tmp: {i};
                rowname(tmp, {'r' + string_of_int(i)});
                v : rowcat(v, tmp);
                i: i + 1;
        rof
        return v;
```

```
fed

#function: sum the rows of a matrix
def double mat sum_row(double mat mx) do
        int x;
        int y;
        double mat s: range(1, cols(mx)) * 0.;
        for x in range(1, (rows(mx))) do
                for y in range(1, (cols(mx))) do
                        s[1, y]: s[1, y] + mx[x, y];
                rof
        rof
        return s;
fed

#function: sum the columns of a matrix
def double mat sum_col(double mat mx) do
        int x;
        int y;
        double mat s: range(1, rows(mx)) * 0.;
        for x in range(1, (cols(mx))) do
                for y in range(1, (rows(mx))) do
                        s[1, y]: s[1, y] + mx[y, x];
                rof
        rof
        return s;
fed

#function: average of the all rows of a matrix

def double mat avg_col(double mat mx) do
        int r;
        int c;
        double sum : 0;
        double avg : 0;
        double mat v;
        for c in range(1, cols(mx)) do
                sum: 0;
                for r in range(1, rows(mx)) do
                        sum : sum + mx[r,c];
                rof
                avg : sum/rows(mx);
                v : colcat(v,{avg});
        rof
        return v;
```

```
fed

#function: average of the all rows of a matrix

def double mat avg_row(double mat mx) do
        int r;
        int c;
        double sum : 0;
        double avg : 0;
        double mat v;
        for r in range(1, rows(mx)) do
                sum: 0;
                for c in range(1, cols(mx)) do
                        sum : sum + mx[r,c];
                rof
                avg : sum/cols(mx);
                v : rowcat(v,{avg});
        rof
        return v;
fed
```

Good Test Cases and Output

```
Good Test Case 0:

# function definition
        disp 'hello world';

Output
hello world


Good Test Case 1:

# test string variable
string endstr;
endstr: 'End\'';
disp endstr;
endstr: 'End\t';
disp endstr;
endstr: 'End\n';
disp endstr;
endstr: 'End\\';
disp endstr;
endstr: 'End"';
```

```
disp endstr;
endstr: 'End\"';
disp endstr;

Output
End'
End
End

End\
End"
End\"
```

Good Test Case 2:

```
# matrix definition
int mat budget: {1,2;3,4};
disp budget;
#budget: {1.1,2.2;3.3,4.4};
#disp budget;

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

Good Test Case 3:

```
#test identifier whether valid
int str_0;
disp str_0;
disp 'str_0 = '+ string_of_int(str_0);

Output
0
str_0 = 0
```

Good Test Case 4:

```
#test seperator
int mat ax: {1, 2; 3, 4};
int i: (2 + 3) * 4;
```

```
disp ax;
disp i;

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

20
```

Good Test Case 5:

```
#test operator
int a;
int b;
int c;
bool x;
bool y;
a:1;
b:2;
c:a+b;
disp c;
x:true;
disp x;
y:1=2;
disp y;
y:1!=2;
disp y;
y:not x;
disp y;
y:(1=2)and true;
disp y;
y:(1=2)or true;
disp y;

Output
3
1
0
1
0
0
1
```

Good Test Case 6:

```
#test data type conversion
int a: 6.6;
double b: 6;
disp a;
disp b;
```

Output

```
6
6
```

Good Test Case 7:

```
#test variable comparation
int i1:1;
int i2:2;
int i3:1;
double d1:1.0;
double d2:2.0;
double d3:1.0;
string s1: 'haha';
string s2: 'hehe';
string s3: 'haha';
bool c;
c:i1=i2;
disp c;
c:i1=i3;
disp c;
c:i1!=i2;
disp c;
c:i1>i2;
disp c;
c:i1>=i2;
disp c;
c:i1<i2;
disp c;
c:i1<=i2;
disp c;
c:d1=d2;
disp c;
c:d1!=d2;
disp c;
```

```
c:d1>d2;
disp c;
c:d1>=d2;
disp c;
c:d1<d2;
disp c;
c:d1<=d2;
disp c;
c:s1=s2;
disp c;
c:s1!=s2;
disp c;
c:s1=s3;
disp c;

Output
0
1
1
0
0
1
1
0
1
0
0
1
1
0
1
1
```

Good Test Case 8:

```
#test operations
string a: 'haha';
string b: 'hehe';
string c: a + b;
string d: 'haha' + 'hehe';
disp c;
disp d;

Output
```

```
hahahehe
hahahehe


Good Test Case 9:

#test if-then-else-fi
int b: 0;
int a: 2;
if b <= 0 then
        disp 0;
else
        disp b;
fi

if a > 0 then
        disp a;
fi

if a < 0 then
else
        disp 0;
fi

Output
0
2
0


Good Test Case 10:

#test if-then-elif-then-else-if
int b: 2;
if b = 0 then
                disp 'b is 0';
elif b = 1 then
                disp 'b is 1';
elif b = 2 then
                disp 'b is 2';
else
                disp 'b not defined';
fi

Output
```

```
b is 2
```

Good Test Case 11:

```
#test counting loop
int b: 0;
int r;
for r in {1;2;3;4} do
        b: b + 1;
        disp b;
rof
```

Output

```
1
2
3
4
```

Good Test Case 12:

```
#test counting loop
import 'buckcal_lib.bc'

int mat b: {1, 2; 3, 4; 5, 6};
int r;
int c;

for r in range(1, rows(b)) do
    for c in range(1, cols(b)) do
        b[r,c]: 0;
    rof
rof

disp b;
```

Output

```
    c1    c2
r1  [0,   0]
r2  [0,   0]
r3  [0,   0]
```

Good Test Case 13:

```
#test conditional loop
int b: 5;
for b <10 do
        b: b + 1;
rof
disp b;
```

Output
10

Good Test Case 14:

```
#test break statement
int r:0;
int a:0;
for r < 100 do
        if a < 4 then
        r: r + 1;
        a: a + 2;
        continue;
        else
        break;
        fi
rof

disp a;
```

Output
4

Good Test Case 15:

```
#test function declaration
def helper(int x);
def helper2(double x);
def int helper3(int);
```

Output

Good Test Case 16:

```
#test function definition
def int helper(int x) do
        disp 'help';
        return 0;
fed
def helper2(double x) do
        x: 0;
        disp x;
fed
```

Output

Good Test Case 17:

```
#test function overloading
def int hello(int a);
def int hello(double a);
def int hello(string a);
```

Output

Good Test Case 18:

```
#test function overloading
def int hello(int a) do
        return a;
fed
def double hello(double a) do
        return a;
fed
def string hello(string a) do
        return a;
fed
```

Output

Good Test Case 19:

```
#test function call
def int returnint (int a) do
                a:a+10;
                return a;
fed

def callreturn (int b) do
                disp b;
                b:b+returnint(b);
                disp b;
fed
int b:4;
callreturn(b);
disp returnint(5);
```

Output
4
18
15

Good Test Case 20:

```
#test string and boolean compare

disp 'h' = 'h';

disp 'a' = 'h';

disp false = false;

disp true = false;
```

Output
1
0
1
0

Good Test Case 21:

```
#test matrix type
```

```
string mat b;
int mat a;
double mat c;
```

Output

Good Test Case 22:

```
#test matrix definition
int mat ax: {1,2,3;4,5,6;7,8,9};
double mat dx: {1.1,2.2;3.3,4.4};
string mat sx : {'ab','cd';'ef','gh'};
disp ax;
disp dx;
disp sx;
```

Output

```
        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
r1      [ab,    cd]
r2      [ef,    gh]
```

Good Test Case 23:

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

Output

```
2
```

Good Test Case 24:

```
#test built-in function rows, cols
int mat ax: {1,2,3;4,5,6;7,8,9};
double mat dx: {1.1,2.2;3.3,4.4};
string mat sx: {'a','b';'c','d';'e','g'};
int a;
a: rows(ax);
disp a;
a: rows(dx);
disp a;
a: rows(sx);
disp a;
a: cols(ax);
disp a;
a: cols(dx);
disp a;
a: cols(sx);
disp a;
```

Output

```
3
2
3
3
2
2
```

Good Test Case 25:

```
#test built-in function string_of_int string_of_double int_of_string
#double_of_string
int a:1;
double b:1.0;
string c;
c:string_of_int(a);
disp c;
c:string_of_double(b);
disp c;
c:'10.1';
a:int_of_string(c);
disp a;
```

```
b:double_of_string(c);
disp b;

Output
1
1
10
10.1


Good Test Case 26:

#test built-in function mat_int_of_string mat_double_of_string
#mat_string_of_int mat_string_of_double mat_int_of_double
#mat_double_of_int
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
        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]
```

Good Test Case 27:

```
#test conditional loop
int b: 0;
int a: 0;
int r;
for b <10 do
        for a < 10 do
                a: a+1;
                r: r+1;
        rof
        a: 0;
        b: b + 1;
rof
disp b;
disp a;
disp r;
```

Output

```
10
0
100
```

Good Test Case 28:

```
#test conditional loop
int a: 0;
int b: 0;
int c: 0;
int r: 0;
for b <10 do
        for a<10 do
                for c<10 do
                if c > 5 then
                        break;
                fi
                r: r+1;
                c: c+1;
                rof
        c:0;
        a: a+1;
        rof
        a:0;
        b: b + 1;
rof
disp b;
disp a;
disp c;
disp r;
```

Output

```
10
0
0
600
```

Good Test Case 29:

```
#test built-in function rows cols
if (cols({1,2,3})=2) then
        disp 'cols=2';
elif (cols({1,2,3,4})=3) then
        disp 'cols=3';
elif (rows({1,2;3,4})=1) then
```

```
        disp 'rows=1';
elif (rows({1,2})=2) then
        disp 'rows=2';
else
        disp 'end';
fi


Output
end


Good Test Case 30:

#test built-in function string_of_int string_of_double int_of_string
#double_of_string
int i;
double d;
string s;
bool b;
i: int_of_string('1.0');
disp i;
d: double_of_string('1.0');
disp d;
s: string_of_int(1.0);
disp s;
s: string_of_double(1.1);
disp s;
b: int_of_string('2')>int_of_string('1');
disp b;
b: double_of_string('22.2')>double_of_string('100.3');
disp b;
b: string_of_int(1)>string_of_int(2);
disp b;


Output
1
1
1
1.1
1
0
0
```

Good Test Case 31:

```
#test built-in function mat_int_of_string mat_double_of_string
#mat_string_of_int mat_string_of_double mat_int_of_double
#mat_double_of_int
int mat ax;
double mat dx;
string mat sx;

ax: mat_int_of_string({'1.1','1.2';'1.3','1.4';'1.5','1.6'});
disp ax;
dx: mat_double_of_string({'1.1','1.2';'1.3','1.4';'1.5','1.6'});
disp dx;
sx: mat_string_of_int({1,2,3;4,5,6;7,8,9});
disp sx;
sx: mat_string_of_double({1.1,2.2;3.3,4.4});
disp sx;
dx: mat_double_of_int(ax);
disp dx;
ax: mat_int_of_double({1.1,2.2;3.3,4.4});
disp ax;
```

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
```

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

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

Good Test Case 32:

```
#test data type conversion
def int cal(double x) do
        int y:x;
        return y;
fed
int i:0;
i: cal(1.1);
disp i;
```

Output

```
1
```

Good Test Case 33:

```
#test data type conversion
def double cal(int x) do
        double a:1.1;
        return 1.1;
fed

int i:0;
i: cal(1);
disp i;
```

Output

```
1
```

Good Test Case 34:

```
#test data type conversion
def double cal(double) do
        return 1+2.2;
```

```
fed
int i:0;
i: cal(1.1);
disp i;
```

Output
3

Good Test Case 35:

```
#test built-in function rowname, colname
int mat ix: {1,2,3; 4,5,6};
double mat dx: {1.1,2.2; 3.3,4.4; 5.5,6.6};
string mat sx: {'a','b','c'; 'd','e','f'};
string mat ssx: {'John','Josh', 'Jay'};
string mat ssx1: {'John','Josh'};
rowname(ix, ssx1);
colname(ix, {'foods','clothes','appliances'});
disp ix;
rowname(dx, {'a','b','c'});
colname(dx, ssx1);
disp dx;
rowname(sx, ssx1);
colname(sx, {'a','b','c'});
disp sx;
```

Output

	foods	clothes	appliances
John	[1,	2,	3]
Josh	[4,	5,	6]

	John	Josh
a	[1.1,	2.2]
b	[3.3,	4.4]
c	[5.5,	6.6]

	a	b	c
John	[a,	b,	c]
Josh	[d,	e,	f]

Good Test Case 36:

```
#test built-in function rowcat, colcat
int mat ix1: {1,2,3;4,5,6};
```

```
int mat ix2: {1,2,3;4,5,6;7,8,9};
int mat ix3: {1,2;3,4};
int mat ix4;
int mat ix5: {7,8,9;10,11,12};
double mat dx1: {1.1,2.2,3.3; 4.4,5.5,6.6};
double mat dx2: {1.1,2.2,3.3;4.4,5.5,6.6;7.7,8.8,9.9};
double mat dx3: {1.1,2.2;3.3,4.4};
double mat dx4;
string mat sx1: {'a','b','c';'d','e','f'};
string mat sx2: {'a','b','c';'d','e','f';'g','h','i'};
string mat sx3: {'a','b';'c','d'};
string mat sx4;
ix4: rowcat(ix1,ix5);
disp ix4;
ix4: colcat(ix1,ix3);
disp ix4;
dx4: rowcat(dx1,dx2);
disp dx4;
dx4: colcat(dx1,dx3);
disp dx4;
sx4: rowcat(sx1,sx2);
disp sx4;
sx4: colcat(sx1,sx3);
disp sx4;
```

Output

```
        c1      c2      c3
r1      [1,     2,      3]
r2      [4,     5,      6]
r1      [7,     8,      9]
r2      [10,    11,     12]

        c1      c2      c3      c1      c2
r1      [1,     2,      3,      1,      2]
r2      [4,     5,      6,      3,      4]

        c1      c2      c3
r1      [1.1,   2.2,    3.3]
r2      [4.4,   5.5,    6.6]
r1      [1.1,   2.2,    3.3]
r2      [4.4,   5.5,    6.6]
r3      [7.7,   8.8,    9.9]

        c1      c2      c3      c1      c2
```

```
r1      [1.1,   2.2,    3.3,    1.1,    2.2]
r2      [4.4,   5.5,    6.6,    3.3,    4.4]

        c1      c2      c3
r1      [a,     b,      c]
r2      [d,     e,      f]
r1      [a,     b,      c]
r2      [d,     e,      f]
r3      [g,     h,      i]

        c1      c2      c3      c1      c2
r1      [a,     b,      c,      a,      b]
r2      [d,     e,      f,      c,      d]
```

Good Test Case 37:

```
#test matrix definition
int mat ax: {1,2,3;4,5,6;7,8,9};
double mat dx: {1.1,2.2;3.3,4.4};
string mat sx : {'ab','cd';'ef','gh'};
int a;
double d;
string s;
a: ax[2,1];
disp a;
a: ax[2,2]+10;
disp a;
a: ax[2,2]+dx[1,1];
disp a;
ax[1,1]:100;
disp ax;
d: dx[1,1]+1;
disp d;
d: dx[1,1]+ax[2,2];
disp d;
dx[1,1]:100.1;
disp dx;
sx[1,1]:'zzz';
disp sx;
```

Output

```
4
15
```

```
6
        c1      c2      c3
r1      [100,   2,      3]
r2      [4,     5,      6]
r3      [7,     8,      9]

2.1
6.1
        c1      c2
r1      [100.1,2.2]
r2      [3.3,   4.4]

        c1      c2
r1      [zzz,   cd]
r2      [ef,    gh]
```

Good Test Case 38:

```
#test abs()
import 'buckcal_lib.bc'

double x : -5;
double y : 5;

disp x;
disp abs(x);
disp abs(y);
```

Output

```
-5
5
5
```

Good Test Case 39:

```
#test range(int x, int y) function and ranger(int x, int y)

import 'buckcal_lib.bc'

double mat mx;

disp range(1,1);
disp range(1,4);
```

```
disp range_col(1, 3);

#output {1}
#output {1, 2, 3, 4}
#output {1; 2; 3}


Output
        c1
r1      [1]

        c1      c2      c3      c4
r1      [1,     2,      3,      4]

        c1
r1      [1]
r2      [2]
r3      [3]
```

Good Test Case 40:

```
#test double mat sum_row(double mat mx) and double mat sum_col(double mat mx)

import 'buckcal_lib.bc'

double mat mx : {1, 2, 3 ; 4, 5, 6; 7, 8, 9; 10, 11, 12};

disp 'sum of rows:';
disp sum_row(mx);

disp 'sum of cols:';
disp sum_col(mx);


Output
sum of rows:
        c1      c2      c3
r1      [22,    26,     30]

sum of cols:
        c1      c2      c3      c4
r1      [6,     15,     24,     33]
```

Good Test Case 41:

```
#test double mat avg_row(double mat mx) and double mat avg_col(double mat mx)

import 'buckcal_lib.bc'

double mat mx : {1, 2, 3 ; 4, 5, 6};

disp 'sum of rows:';
disp sum_row(mx);
disp 'sum of cols:';
disp sum_col(mx);
```

Output

```
sum of rows:
        c1      c2      c3
r1      [5,     7,      9]

sum of cols:
        c1      c2
r1      [6,     15]
```

Good Test Case 42:

```
#test built-in function abs
import 'buckcal_lib.bc'
double x:1.1;
disp abs(x);
x:-1.1;
disp abs(x);
```

Output

```
1.1
1.1
```

Good Test Case 43:

```
#test built-in function range
import 'buckcal_lib.bc'
int i;
```

```
for i in range(1,4) do
        disp 'hello';
rof

Output
hello
hello
hello
hello

Good Test Case 44:

#test built-in function sum_row
import 'buckcal_lib.bc'
int mat ix:{1,2,3 ; 4,5,6 ; 7,8,9};
double mat dx: {1.1,2.2,3.3;4.4,5.5,6.6;7.7,8.8,9.9};
double mat result:sum_row(ix);
disp result;
result:sum_row(dx);
disp result;
result:sum_row({10.1,20.1;30.1,40.1});
disp result;

Output
        c1      c2      c3
r1      [12,    15,     18]

        c1      c2      c3
r1      [13.2,  16.5,   19.8]

        c1      c2
r1      [40.2,  60.2]



Good Test Case 45:

# and or, not

bool t: true;
bool f: false;

disp (t and f);
disp (f and t);
disp (t and t);
```

```
disp (f and f);

disp (t or f);
disp (f or t);
disp (t or t);
disp (f or f);

disp (not t);
disp (not f);

disp (f and t or f);
disp (f and (t or f));
disp (not f and t);
disp (not f or t);

Output
0
0
1
0
1
1
1
0
0
1
0
0
1
1
```

Good Test Case 46:

```
#test built-in function sum_col
import 'buckcal_lib.bc'
int mat ix:{1,2,3 ; 4,5,6 ; 7,8,9};
double mat dx: {1.1,2.2,3.3;4.4,5.5,6.6;7.7,8.8,9.9};
double mat result:sum_col(ix);
disp result;
result:sum_col(dx);
disp result;
result:sum_col({10.1,20.1;30.1,40.1});
disp result;
```

```
Output
        c1      c2      c3
r1      [6,     15,     24]

        c1      c2      c3
r1      [6.6,   16.5,   26.4]

        c1      c2
r1      [30.2,  70.2]
```

Good Test Case 47:

```
#test built-in function avg_row
import 'buckcal_lib.bc'
int mat ix:{1,2,3 ; 4,5,6 ; 7,8,9};
double mat dx: {1.1,2.2,3.3;4.4,5.5,6.6;7.7,8.8,9.9};
double mat result:avg_row(ix);
disp result;
result:avg_row(dx);
disp result;
result:avg_row({10.1,20.1;30.1,40.1});
disp result;
```

```
Output
        c1
r1      [2]
r1      [5]
r1      [8]

        c1
r1      [2.2]
r1      [5.5]
r1      [8.8]

        c1
r1      [15.1]
r1      [35.1]
```

Good Test Case 48:

```
#test built-in function avg_col
import 'buckcal_lib.bc'
int mat ix:{1,2,3 ; 4,5,6 ; 7,8,9};
double mat dx: {1.1,2.2,3.3;4.4,5.5,6.6;7.7,8.8,9.9};
double mat result:avg_col(ix);
disp result;
result:avg_col(dx);
disp result;
result:avg_col({10.1,20.1;30.1,40.1});
disp result;


Output
        c1      c1      c1
r1      [4,     5,      6]

        c1      c1      c1
r1      [4.4,   5.5,    6.6]

        c1      c1
r1      [20.1,  30.1]



Good Test Case 49:

# test built-in function strlen
string sx: 'Hello world';
int ix:strlen(sx);
disp sx;
disp ix;

Output
Hello world
11


Good Test Case 50:

# test built-in function slice
string sx:'helloworld';
int ix;
ix:strlen(sx);
disp sx;
```

```
disp ix;
sx:slice(sx,2,5);
ix:strlen(sx);
disp sx;
disp ix;

Output
helloworld
10
loworl
6


Good Test Case 51:

# test built-in function getrow
int mat ix:{1,2,3;4,5,6;7,8,9};
double mat dx:{1.1,2.2,3.3;4.4,5.5,6.6;7.7,8.8,9.9};
string mat sx:{'a','b','c';'d','e','f';'i','j','k'};
int mat ix2:getrow(ix,2);
double mat dx2:getrow(dx,2);
string mat sx2:getrow(sx,2);
disp ix2;
disp dx2;
disp sx2;


Output
        c1      c2      c3
r2      [4,     5,      6]

        c1      c2      c3
r2      [4.4,   5.5,    6.6]

        c1      c2      c3
r2      [d,     e,      f]



Good Test Case 52:

# test built-in function getcol
int mat ix:{1,2,3;4,5,6;7,8,9};
double mat dx:{1.1,2.2,3.3;4.4,5.5,6.6;7.7,8.8,9.9};
string mat sx:{'a','b','c';'d','e','f';'i','j','k'};
```

```
int mat ix2:getcol(ix,2);
double mat dx2:getcol(dx,2);
string mat sx2:getcol(sx,2);
disp ix2;
disp dx2;
disp sx2;


Output
        c2
r1      [2]
r2      [5]
r3      [8]

        c2
r1      [2.2]
r2      [5.5]
r3      [8.8]

        c2
r1      [b]
r2      [e]
r3      [j]



Good Test Case 53:

# test built-in function setrow
int mat ix1:{1,2,3;4,5,6;6,7,8};
int mat ix2:{1,2,3};
double mat dx1:{1.1,2.2,3.3;4.4,5.5,6.6;7.7,8.8,9.9};
double mat dx2:{1.1,2.2,3.3};
string mat sx1:{'a','b','c';'d','e','f';'i','j','k'};
string mat sx2:{'a','b','c'};
setrow(ix1, 2, ix2);
setrow(dx1, 2, dx2);
setrow(sx1, 2, sx2);
disp ix1;
disp dx1;
disp sx1;

Output
        c1      c2      c3
r1      [1,     2,      3]
```

```
r1      [1,     2,      3]
r3      [6,     7,      8]

        c1      c2      c3
r1      [1.1,   2.2,    3.3]
r1      [1.1,   2.2,    3.3]
r3      [7.7,   8.8,    9.9]

        c1      c2      c3
r1      [a,     b,      c]
r1      [a,     b,      c]
r3      [i,     j,      k]
```

Good Test Case 54:

```
# test built-in function setcol
int mat ix1:{1,2,3;4,5,6;6,7,8};
int mat ix2:{1;2;3};
double mat dx1:{1.1,2.2,3.3;4.4,5.5,6.6;7.7,8.8,9.9};
double mat dx2:{1.1;2.2;3.3};
string mat sx1:{'a','b','c';'d','e','f';'i','j','k'};
string mat sx2:{'a';'b';'c'};
setcol(ix1, 2, ix2);
setcol(dx1, 2, dx2);
setcol(sx1, 2, sx2);
disp ix1;
disp dx1;
disp sx1;
```

Output

```
        c1      c1      c3
r1      [1,     1,      3]
r2      [4,     2,      6]
r3      [6,     3,      8]

        c1      c1      c3
r1      [1.1,   1.1,    3.3]
r2      [4.4,   2.2,    6.6]
r3      [7.7,   3.3,    9.9]

        c1      c1      c3
r1      [a,     a,      c]
```

```
r2      [d,     b,      f]
r3      [i,     c,      k]
```

Good Test Case 55:

```
# test built-in function init_mat
int mat ix: init_mat(2,3,1);
double mat dx:init_mat(4,3,4.4);
string mat sx:init_mat(5,6,'a');
disp ix;
disp dx;
disp sx;
```

Output

```
        c1      c2      c3
r1      [1,     1,      1]
r2      [1,     1,      1]

        c1      c2      c3
r1      [4.4,   4.4,    4.4]
r2      [4.4,   4.4,    4.4]
r3      [4.4,   4.4,    4.4]
r4      [4.4,   4.4,    4.4]

        c1      c2      c3      c4      c5      c6
r1      [a,     a,      a,      a,      a,      a]
r2      [a,     a,      a,      a,      a,      a]
r3      [a,     a,      a,      a,      a,      a]
r4      [a,     a,      a,      a,      a,      a]
r5      [a,     a,      a,      a,      a,      a]
```

Good Test Cast 56:

```
import 'import1.bc'
import 'import2.bc'
a();
b();
```

import1.bc

```
import 'import11.bc'
import 'import12.bc'
import 'import_common.bc'

def a() do
a1();
a2();
common();
fed
```

import11.bc

```
def a1() do
disp 'a1';
fed
```

import12.bc

```
def a2() do
disp 'a2';
fed
```

import2.bc

```
import 'import_common.bc'

def b() do
disp 'b';
common();
fed
```

import_common.bc

```
def common() do
disp 'common';
fed

Output:
a1
a2
```

```
common
b
common
```

Bad Test Cases and Output

Bad Test Case 0:

```
# display int
=disp 'Hello World';
```

Output
Parser error: token =, in sample0.bc line 2,1

Bad Test Case 1:

```
# test string variable
string endstr;
endstr: 'End'';
disp endstr;
```

Output
Scanner error: Unexpected End-of-Line

Bad Test Case 2:

```
# matrix definition
mat int budget: {1,2;3,4};
disp budget;
```

Output
Parser error: token int, in sample2.bc line 2,5

Bad Test Case 3:

```
#test Keywords import
int import;
```

Output
Parser error: token import, in sample3.bc line 2,5

Bad Test Case 4:

```
#test identifier whether valid
int str*;
```

Output

```
Parser error: token *, in sample4.bc line 2,8
```

Bad Test Case 5:

```
#test seperator
int mat ax: {1, 2; 3, 4};
int i: {2 + 3} * 4;
```

Output

```
Sast error: variable and expression type mismatch
```

Bad Test Case 6:

```
#test operator
bool x;
bool y;
x:true;
y:1 not 2;
disp y;
```

Output

```
Parser error: token not, in sample6.bc line 5,5
```

Bad Test Case 7:

```
#test operator
int a;
a:=2;
disp a;
```

Output

```
Parser error: token =, in sample7.bc line 3,3
```

Bad Test Case 8:

```
#test variable comparation
string s1: 'haha';
int i1: 1;
```

```
bool c;
c:s1>i1;
disp c;
Output
Sast error: ">" bad operand type


Bad Test Case 9:

#test variable comparation
int i1:1;
string s2: 'hehe';
bool c;
c:i1=s2;
disp c;
c:i1>s2;
disp c;
Output
Sast error: "=" bad operand type


Bad Test Case 10:

#test operations
string a: 'haha';
string b: 'hehe';
string d: 'haha' - 'hehe';
disp d;
Output
Sast error: "-" bad operand type


Bad Test Case 11:

#test if-then-else-fi
int b: 0;
int a: 2;
if b <= 0 then
        disp 0;
else
        disp b;

Output
Parser error: token , in sample11.bc line 8,1
```

Bad Test Case 12:

```
#test if-then-elif-then-else-if
int b: 2;
if b = 0 then
		disp 'b is 0';
elif b = 1 then
		disp 'b is 1';
else b = 2 then
		disp 'b is 2';
else
		disp 'b not defined';
fi
```

Output

Parser error: token then, in sample12.bc line 7,12

Bad Test Case 13:

```
#test if-then-elif-then-else-if
int b: 2;
if b = 0 then
		disp 'b is 0';
fi
elif b = 1 then
		disp 'b is 1';
else
		disp 'b is 2';
fi
```

Output

Parser error: token elif, in sample13.bc line 6,1

Bad Test Case 14:

```
#test if-then-elif-then-else-if
int b: 2;
if b = 0 then
		disp 'b is 0';
else
fi
elif b = 1 then
		disp 'b is 1';
```

Output

```
Parser error: token elif, in sample14.bc line 7,1


Bad Test Case 15:

#test counting loop
int b: 0;
int r;
for r in [1;2;3;4] do
        b: b + 1;
        disp b;
rof
Output
Parser error: token [, in sample15.bc line 4,10


Bad Test Case 16:

#test counting loop
int mat b;
int r;
b[r,0]: 1;

for r in range(1, rows(b)) do
        b[r,0]: 1;
rof
Output
Sast error: function range not defined


Bad Test Case 17:

#test conditional loop
int b: 5;
for b <10 do
        b: b + 1;
disp b;
Output
Parser error: token , in sample17.bc line 5,8


Bad Test Case 18:

#test break continue statement
int a:0;
```

```
if a < 100 then
else
        a: a + 50;
        continue;
fi
```

Output
Sast error: Continue should only be used inside a loop

Bad Test Case 19:

```
#test break continue statement
int a:0;
if a < 100 then
else
        a: a + 50;
        break;
fi
```

Output
Sast error: Break should only be used inside a loop

Bad Test Case 20:

```
#test function declaration
helper(int x);
```

Output
Parser error: token int, in sample20.bc line 2,8

Bad Test Case 21:

```
#test function definition
def int helper() do
        string a: 'a';
        disp 'help';
        return a;
fed
```

Output
Sast error: mismatch with function's return type

Bad Test Case 22:

```
#test function definition
def int helper() do
	disp 'help';
fed
```

Output
Sast error: Function 'helper' return statement missing

Bad Test Case 23:

```
#test function definition
def helper() do
	string a: 'a';
	disp 'help';
	return a;
fed
```

Output
Sast error: mismatch with function's return type

Bad Test Case 24:

```
#test function overloading
def int hello(int a);
def hello(int);
```

Output
Sast error: Function 'hello' already defined

Bad Test Case 25:

```
#test function overloading
def int hello(int a) do
	return a;
fed
def hello(int) do
	disp 'hello';
fed
```

Output
Sast error: Function 'hello' already defined

Bad Test Case 26:

```
#test function call
def returnint (int a) do
		a:a+10;
fed

def callreturn (int b) do
		disp b;
		b:returnint(b);
		disp b;
fed
int b:4;
callreturn(b);
```

Output

Sast error: Int : Void operand types invalid

Bad Test Case 27:

```
#test matrix type
bool mat b;
```

Output

Parser error: token b, in sample27.bc line 2,10

Bad Test Case 28:

```
#test matrix definition
int mat ax: {'ab','cd';'ef','gh'};
disp ax;
```

Output

Sast error: variable and expression type mismatch

Bad Test Case 29:

```
#test matrix definition
int mat ax: [1,2;3,4];
disp ax;
```

Output

Parser error: token [, in sample29.bc line 2,13

Bad Test Case 30:

```
#test function overloading
def int hello(int a);
def hello(int) do
        disp 'hello';
fed
Output
Sast error: Function 'hello' return type different with declaration
```

Bad Test Case 31:

```
#test function overloading
def int hello(int) do
        disp 'hello';
fed
def int hello(int a);
Output
Sast error: Function 'hello' return statement missing
```

Bad Test Case 32:

```
#test conditional loop
int a: 0;
int b: 0;
int c: 0;
int r: 0;
for b <10 do
        for a<10 do
                for c<10 do
                r: r+1;
                c: c+1;
                #rof
        c:0;
        a: a+1;
        rof
        a:0;
        b: b + 1;
rof
disp b;
disp a;
disp c;
disp r;

Output
```

```
Parser error: token , in sample32.bc line 22,1
```

Bad Test Case 33:

```
#test matrix definition
int mat ax: {1,2;3,4,5};
disp ax;
Output
Sast error: Mat rows must have same length
```

Bad Test Case 34:

```
#test function call
def int cal(double) do
        disp 1;
        return 1;
fed

int i;
i: cal();
Output
Sast error: function cal not defined
```

Bad Test Case 35:

```
#test function call
def int cal(double, int) do
        disp 1;
        return 1;
fed

int i;
i: cal('a', i);
Output
Sast error: function cal not defined
```

Bad Test Case 36:

```
#test function call
def int cal(double a, double b) do
        disp 1;
```

```
        return 1;
fed

double i;
i: cal(i,i,i);
```

Output
Sast error: function cal not defined

Bad Test Case 37:

```
#test built-in function rowname, colname
int mat ix: {1,2,3;4,5,6};
double mat dx: {1.1,2.2;3.3,4.4;5.5,6.6};
string mat sx: {'a','b','c';'d','e','f'};
string mat ssx: {'John','Josh'};
rowname(ix, {'foods','clothes','appliances'});
colname(ix, ssx);
disp ix;
rowname(dx, {'a','b','c';'a','b','c'});
colname(dx, ssx);
disp dx;
rowname(sx, {'a','b','c'});
colname(sx, {'a','b','c'});
disp sx;
```

Output
rowname: name matrix does not have enough entries

Bad Test Case 38:

```
#test built-in function rowname, colname
int mat ix: {1,2,3;4,5,6};
double mat dx: {1.1,2.2;3.3,4.4;5.5,6.6};
string mat sx: {'a','b','c';'d','e','f'};
string mat ssx: {'John','Josh'};
rowname(ix, {1,2});
colname(ix, {'foods','clothes','appliances'});
disp ix;
rowname(dx, {1.1,2.2,3.3});
colname(dx, ssx);
disp dx;
rowname(sx, ssx);
colname(sx, {1,2,3});
```

```
disp sx;
```

Output
Sast error: function rowname not defined

Bad Test Case 39:

```
#test built-in function rowname, colname
int mat ix: {1,2,3;4,5,6};
double mat dx: {1.1,2.2;3.3,4.4;5.5,6.6};
string mat sx: {'a','b','c';'d','e','f'};
string mat ssx: {'John','Josh'};
rowname(ix, 'a');
colname(ix, {'foods','clothes','appliances'});
disp ix;
rowname(dx, 1);
colname(dx, ssx);
disp dx;
rowname(sx, ssx);
colname(sx, 2.2);
disp sx;
```

Output
Sast error: function rowname not defined

Bad Test Case 40:

```
#test built-in function rowcat, colcat
int mat ix1: {1,2,3;4,5,6};
int mat ix2: {1,2,3;4,5,6;7,8,9};
int mat ix3: {1,2;3,4};
int mat ix4;
double mat dx1: {1.1,2.2;3.3,4.4;5.5,6.6};
double mat dx2: {1.1,2.2,3.3;4.4,5.5,6.6;7.7,8.8,9.9};
double mat dx3: {1.1,2.2;3.3,4.4};
double mat dx4;
string mat sx1: {'a','b','c';'d','e','f'};
string mat sx2: {'a','b','c';'d','e','f';'g','h','i'};
string mat sx3: {'a','b';'c','d'};
string mat sx4;
ix4: rowcat(ix1,ix3);
disp ix4;
ix4: colcat(ix1,ix3);
```

```
disp ix4;
dx4: rowcat(dx1,dx2);
disp dx4;
dx4: colcat(dx1,dx2);
disp dx4;
sx4: rowcat(sx1,sx2);
disp sx4;
sx4: colcat(sx1,sx2);
disp sx4;

Output
rowcat: matrix does not have the same number of columns


Bad Test Case 41:

#test built-in function rowcat, colcat
int mat ix1: {1,2,3;4,5,6};
int mat ix2: {1,2,3;4,5,6;7,8,9};
int mat ix3: {1,2;3,4};
int mat ix4;
double mat dx1: {1.1,2.2;3.3,4.4;5.5,6.6};
double mat dx2: {1.1,2.2,3.3;4.4,5.5,6.6;7.7,8.8,9.9};
double mat dx3: {1.1,2.2;3.3,4.4};
double mat dx4;
string mat sx1: {'a','b','c';'d','e','f'};
string mat sx2: {'a','b','c';'d','e','f';'g','h','i'};
string mat sx3: {'a','b';'c','d'};
string mat sx4;
ix4: rowcat(ix1,dx2);
disp ix4;
ix4: colcat(ix1,sx3);
disp ix4;


Output
Sast error: function colcat not defined


Bad Test Case 42:

#test data type conversion
def int cal(double x) do
        return x;
fed
```

```
int i:0;
i: cal(1.1);
Output
Sast error: mismatch with function's return type


Bad Test Case 43:

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

Output
matsub: index check failed, bad index


Bad Test Case 44:

#test matrix definition
double mat dx: {1.1,2.2;3.3,4.4};
double d;
d: dx[3,1]+1;
disp d;

Output
matsub: index check failed, bad index


Bad Test Case 45:

#test matrix definition
string mat sx : {'ab','cd';'ef','gh'};
string s;
s: sx[1,3];
disp s;
Output
matsub: index check failed, bad index


Bad Test Case 46:

#test matrix definition
string mat sx : {'ab','cd';'ef','gh'};
```

```
string s;
sx[1,1]:1;
disp s;
```

Output
Sast error: String : Int operand types invalid

Bad Test Case 47:

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

Output
Sast error: Int : String operand types invalid

Bad Test Case 48:

```
#test matrix definition
double mat dx: {1.1,2.2;3.3,4.4};
double d;
dx[1,1]:'a';
disp dx;
```

Output
Sast error: Double : String operand types invalid

Bad Test Case 49:

```
#test abs()
import 'buckcal_lib.bc'

double bad_int : -0;

disp abs(bad_int);

double bad_int2: -1;

disp "the integer:"
disp bad_int2;

disp "avg of the integer:"
```

```
disp abs(bad_int2);
Output
Parser error: token double, in sample49.bc line 8,1


Bad Test Case 50:

#test range(int x, int y) function and ranger(int x, int y)

import 'buckcal_lib.bc'

range(1,-3);

Output


Bad Test Case 51:

#test double mat sum_row(double mat mx) and double mat sum_col(double mat mx)

import 'buckcal_lib.bc'

double mat mx : {1, 2, 3 ; 4, 5, 6; 7, 8, 9; 10, 11, 12};

disp "sum of rows:"
disp sum_row(mx);

disp "sum of cols:"
disp sum_col(mx);
Output
Scanner error: illegal character ", in sample51.bc line 7,6


Bad Test Case 52:

#test double mat avg_row(double mat mx) and double mat avg_col(double mat mx)

import 'buckcal_lib.bc'

double mat mx : {1, 2, 3 ; 4, 5, 6};

disp "avg of rows:";
disp sum_row(mx);

disp "avg of cols:"
```

```
disp sum_col(mx);
Output
Scanner error: illegal character '', in sample52.bc line 7,6


Bad Test Case 53:

#test built-in function abs
disp abs('a');
Output
Sast error: function abs not defined


Bad Test Case 54:

#test built-in function range
int i;
for i in range(1,4.5) do
        disp 'hello';
rof

for i in range(1,'a') do
        disp 'hello';
rof
Output
Sast error: function range not defined


Bad Test Case 55:

#test built-in function range
int i;
for i in range[1,4] do
        disp 'hello';
rof
Output
Sast error: variable range not defined


Bad Test Case 56:

#test built-in function range
int i;
for i in range{1,4} do
        disp 'hello';
```

```
rof
Output
Parser error: token {, in sample56.bc line 3,15


Bad Test Case 57:

#test built-in function sum_row
double mat result:sum_row({1,2,3 ; 4,5,6 ; 7,8});
disp mat;

Output
Sast error: Mat rows must have same length


Bad Test Case 58:

#test built-in function sum_row
string mat result:sum_col({1,2,3 ; 4,5,6 ; 7,8,9});
disp mat;

Output
Sast error: function sum_col not defined


Bad Test Case 59:

#test built-in function sum_col
double mat result:sum_col({1,2,3 ; 4,5,6 ; 7,8});
disp mat;

Output
Sast error: Mat rows must have same length


Bad Test Case 60:

#test built-in function sum_col
string mat result:sum_col({1,2,3 ; 4,5,6 ; 7,8,9});
disp mat;
Output
Sast error: function sum_col not defined


Bad Test Case 61:
```

```
#test built-in function avg_row
double mat result:avg_row({1,2,3 ; 4,5,6 ; 7,8});
disp mat;
Output
Sast error: Mat rows must have same length
```

Bad Test Case 62:

```
#test built-in function avg_row
string mat result:avg_row({1,2,3 ; 4,5,6 ; 7,8,9});
disp mat;
Output
Sast error: function avg_row not defined
```

Bad Test Case 63:

```
#test built-in function avg_col
double mat result:avg_col({1,2,3 ; 4,5,6 ; 7,8});
disp mat;
Output
Sast error: Mat rows must have same length
```

Bad Test Case 64:

```
#test built-in function avg_col
import 'buckcal_lib.bc'
string mat result:avg_row({1,2,3 ; 4,5,6 ; 7,8,9});
disp mat;
Output
Sast error: variable and expression type mismatch
```

Bad Test Case 65:

```
# test built-in function strlen
disp strlen(123);
Output
Sast error: function strlen not defined
```

Bad Test Case 66:

```
# test built-in function slice
string sx:"helloworld";
sx:slice(sx,2,20);
disp sx;
Output
Scanner error: illegal character ", in sample66.bc line 2,11
```

Bad Test Case 67:

```
# test built-in function slice
string sx:slice(1234,2,5);
disp sx;
Output
Sast error: function slice not defined
```

Bad Test Case 68:

```
# test built-in function getrow
int mat ix:{1,2,3;4,5,6;7,8,9};
int mat ix2:getrow(ix,5);
disp ix2;
Output
getrow: argument not in row range
```

Bad Test Case 69:

```
# test built-in function getcol
int mat ix:{1,2,3;4,5,6;7,8,9};
int mat ix2:getcol(ix,5);
disp ix2;
Output
getcol: argument not in col range
```

Bad Test Case 70:

```
# test built-in function setrow
int mat ix1:{1,2,3;4,5,6;6,7,8};
int mat ix2:{1,2,3,4};
setrow(ix1, 2, ix2);
disp ix1;
```

```
Output
setrow: matrix to be set does not have matching column number


Bad Test Case 71:

# test built-in function setcol
int mat ix1:{1,2,3;4,5,6;6,7,8};
int mat ix2:{1,2,3,4};
setcol(ix1, 2, ix2);
disp ix1;
Output
setcol: matrix to be set does not have matching row number


Bad Test Case 72:

# test built-in function init_mat
int mat ix: init_mat(2,3,4.4);
double mat dx:init_mat(4,3,4.4);
string mat sx:init_mat(5,6,1);
disp ix;
disp dx;
disp sx;
Output
Sast error: variable and expression type mismatch
```

User Sample Program

sample.bc

```
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 sbudget;
double mat tmp;

# initialize
budget: {1+1, 3.3};
colname(budget, {'Food', 'Cola'});
rowname(budget, {'John'});
# add one column and naming
tmp : {0.1};
colname(tmp, {'Paper'});
budget: colcat(budget, tmp);
# add one row with naming
budget: addrow(budget, {1*2, 0.7, 5.10}, {'Tom'});
# add sum row
budget: addrow(budget, sum_row(budget), {'sum'});
# element access by sub matrix expression
budget[3, 2]: gcd(15, 20);
# display
disp budget;
sbudget: {5; 8; 14};
rowname(sbudget, {'John', 'Tom', 'sum'});
colname(sbudget, {'Balance'});
```

```
disp sbudget;
tmp: gauss_elim(budget, sbudget);
rowname(tmp, {'Food', 'Cola', 'Paper'});
colname(tmp, {'Price'});
disp tmp;
```

imp.bc

```
import 'buckcal_lib.bc'

def double mat trans(double mat x) do
 double mat ret;
 int i; int j;
 ret : init_mat(cols(x), rows(x), 0.0);
 for i in range(1, rows(x)) do
  for j in range(1, cols(x)) do
   ret[j, i] : x[i, j];
  rof
 rof
 return ret;
fed

def int mat range_inv(int x, int y) do
        int mat v;
        int i : x;
 int j : 1;
        int mat tmp;
```

```
        for (i >= y) do
                tmp: {i};
                colname(tmp, {'c' + string_of_int(j)});
                v : colcat(v, tmp);
                i: i - 1;
  j: j + 1;
        rof
        return v;
fed



# return x%y
def int mod(int x, int y) do
        int c : x / y;
        return x - y*c;
fed

# Euclidean Greatest Common Divisor
def int gcd(int a, int b) do
  int c;
  # m: a; n: b;
  for (b!=0) do  #Remainder is not zero
  c: mod(a, b); a: b; b: c; #continue to divide until the remainder is 0
  rof
  return a;
fed
```

```
def double mat gauss_elim(double mat a, double mat b) do
  int i; int j; int k; int n;
  double c; double sum;
  double mat x;

  n: rows(a);
  x: range_col(1, n) * 1.2;
  # loop for the generation of upper triangular matrix
  for i in range(1, n-1) do
    for j in range(1, n) do
      if j > i then
          c : a[j, i] / a[i, i];
          for k in range(i, n) do
            a[j, k] : a[j, k] - c * a[i, k];
          rof
          b[j, 1] : b[j, 1] - c * b[i, 1];
        fi
    rof
  rof

  # this loop is for backward substitution
  x[n, 1] : b[n, 1] / a[n, n];
  for i in range_inv(n-1, 1) do
    sum: b[i, 1];
    for j in range(i+1, n) do
      sum: sum - a[i, j]*x[j, 1];
```

```
    rof
    x[i, 1] : sum / a[i, i];
  rof
  return x;
fed
```

Scripts

test.sh

```
#! /bin/bash

# this script run all the test samples
# ./test.sh <path to compiler>

# command shorthand
TRANS=$1
NU=/dev/null
DIF="diff -b -B"
CPPC=g++

# path setting
GOOD_DIR=test_samples_good
BAD_DIR=test_samples_bad
IDEAL_DIR=test_ideal_outputs
OUTPUT_DIR=outputs
CFILE_DIR=cfiles
BINFILE=binfiles

# object files
CLIBOBJ="buckcal_mat.o buckcal_core.o"

# checking argument
if [ "$1" == "" ]; then
        echo "Error: missing path of compiler"
        exit 1
fi

if [ ! -f $1 ]; then
```

```
        echo "Error: compiler does not exist"
        exit 1
fi

# mkdir for testing
mkdir $OUTPUT_DIR $CFILE_DIR $BINFILE 2> $NU

# copy C++ library here
cp ../lib/buckcal_mat.cpp $CFILE_DIR
cp ../lib/buckcal_mat.hpp $CFILE_DIR
cp ../lib/buckcal_core.cpp $CFILE_DIR

# compile C++ library
$CPPC -c $CFILE_DIR/buckcal_core.cpp -o $CFILE_DIR/buckcal_core.o
$CPPC -c $CFILE_DIR/buckcal_mat.cpp -o $CFILE_DIR/buckcal_mat.o

# run good cases
for (( i =  0; i <= 55; i++ )) do
        rm -rf $CFILE_DIR/goodsample${i}
        mkdir $CFILE_DIR/goodsample${i} 2> $NU
        cd $CFILE_DIR/goodsample${i}
        cp ../../$GOOD_DIR/sample${i}.bc ./
        cp ../buckcal_core.o ./
        cp ../buckcal_mat.o ./
        cp ../buckcal_mat.hpp ./
        cp ../../../lib/buckcal_lib.bc ./
        ../../$TRANS sample${i}.bc goodsample${i}.cpp 2>
../../$OUTPUT_DIR/goodsample${i}out.txt
        if [ -s ../../$OUTPUT_DIR/goodsample${i}out.txt ]; then
                echo "Good sample${i}.bc error: compiler should not output any message"
        fi
        if [ -s goodsample${i}.cpp ]; then
                $CPPC -w $CLIBOBJ *.cpp -o ../../$BINFILE/goodsample${i}.bin \
                2>> ../../$OUTPUT_DIR/goodsample${i}out.txt
                ../../$BINFILE/goodsample${i}.bin 1>>
../../$OUTPUT_DIR/goodsample${i}out.txt
        fi
        $DIF ../../$IDEAL_DIR/goodsample${i}ideal.txt
../../$OUTPUT_DIR/goodsample${i}out.txt > $NU
        if [ $? -eq 1 ]; then
                echo "Good sample${i}.bc error: program output does not match ideal one"
        fi
        cd ../../
done
```

```
# import test case
rm -rf $CFILE_DIR/goodsample56
cp -r $GOOD_DIR/sample56 $CFILE_DIR
mv $CFILE_DIR/sample56 $CFILE_DIR/goodsample56
cd $CFILE_DIR/goodsample56
cp ../buckcal_core.o ./
cp ../buckcal_mat.o ./
cp ../buckcal_mat.hpp ./
cp ../../../lib/buckcal_lib.bc ./
../../$TRANS sample56.bc goodsample56.cpp 2>
../../$OUTPUT_DIR/goodsample56out.txt
if [ -s ../../$OUTPUT_DIR/goodsample56out.txt ]; then
        echo "Good sample56.bc error: compiler should not output any message"
fi
if [ -s goodsample56.cpp ]; then
        $CPPC -w $CLIBOBJ *.cpp -o ../../$BINFILE/goodsample56.bin \
        2>> ../../$OUTPUT_DIR/goodsample56out.txt
        ../../$BINFILE/goodsample56.bin 1>> ../../$OUTPUT_DIR/goodsample56out.txt
fi
$DIF ../../$IDEAL_DIR/goodsample56ideal.txt ../../$OUTPUT_DIR/goodsample56out.txt >
$NU
if [ $? -eq 1 ]; then
        echo "Good sample56.bc error: program output does not match ideal one"
fi
cd ../../

echo "Good sample test done"

# run bad cases
for (( i =  0; i <= 72; i++ )) do
        rm -rf $CFILE_DIR/badsample${i}
        mkdir $CFILE_DIR/badsample${i} 2> $NU
        cd $CFILE_DIR/badsample${i}
        cp ../../$BAD_DIR/sample${i}.bc ./
        cp ../buckcal_core.o ./
        cp ../buckcal_mat.o ./
        cp ../buckcal_mat.hpp ./
        cp ../../../lib/buckcal_lib.bc ./
        ../../$TRANS sample${i}.bc badsample${i}.cpp 2>
../../$OUTPUT_DIR/badsample${i}out.txt
        if [ -s badsample${i}.cpp ]; then
                $CPPC -w $CLIBOBJ *.cpp -o ../../$BINFILE/badsample${i}.bin \
                2>> ../../$OUTPUT_DIR/badsample${i}out.txt
                ../../$BINFILE/badsample${i}.bin 2>>
../../$OUTPUT_DIR/badsample${i}out.txt
```

```
        fi
        $DIF ../../$IDEAL_DIR/badsample${i}ideal.txt
../../$OUTPUT_DIR/badsample${i}out.txt > $NU
        if [ $? -eq 1 ]; then
                echo "Bad sample${i}.bc error: compiler output message does not match
ideal one"
        fi
        cd ../../
done
echo "Bad sample test done"
```

compile.sh

```
#! /bin/bash

# this script compile a file for user
# ./compile.sh <main buckcal source file> <output executable name>

# command shorthand
NU=/dev/null
CPPC=g++

# path setting
COMPILER=../src/main.bin

# checking and file

if [ ! -f $COMPILER ]; then
        echo "Error: compiler not found, please make first"
        exit 1
fi

if [ "$1" == "" ]; then
        echo "Error: missing path of input file"
        exit 1
fi

if [ ! -f $1 ]; then
        echo "Error: input file does not exist"
        exit 1
fi

if [ "$2" == "" ]; then
        echo "Error: missing path of output executable"
        exit 1
fi
```

```
# copy files to work folder
cp ../lib/buckcal_mat.cpp ./
cp ../lib/buckcal_mat.hpp ./
cp ../lib/buckcal_core.cpp ./
cp ../lib/buckcal_lib.bc ./

# compile
$COMPILER $1 $1.cpp

if [ ! -s $1.cpp ]; then
        echo "Error: cannot compile $1"
        exit 1
fi

$CPPC -w *.cpp -o $2

if [ ! -f $2 ]; then
        echo "Error: cannot generate executable"
        exit 1
fi

echo "Compile complete, $2 is ready"
```

Appendix C – Project Log

12/17/14	Lingyuan He	On no, not again…
12/17/14	Lingyuan He	Well, that was too soon, now introducing the last commit!
12/17/14	Lingyuan He	fix Makefile clean and test.sh
12/17/14	Lingyuan He	remove PWD in scripts
12/17/14	Lingyuan He	merge back dynamic_link for a better import
12/17/14	Lingyuan He	sample 56 enhanced
12/17/14	Lingyuan He	usr folder and sample
12/17/14	Lingyuan He	import specific case
12/17/14	Lingyuan He	Merge branch 'dynamic_link' of https://github.com/Maruf789/PLT into d…
12/17/14	Lingyuan He	fix bool=bool compare, allow tests to proceed with new link mechanism
12/17/14	wmcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/17/14	wmcu	main extry
12/17/14	wmcu	formatting
12/16/14	wmcu	catch parser exception
12/16/14	Ahmad Maruf	OMG Last Commit! Yay!
12/16/14	wmcu	import done
12/16/14	wmcu	scheck default import lib_funs
12/15/14	wmcu	merge
12/15/14	Ahmad Maruf	usr folder
12/15/14	wmcu	remove sample.bin in make clean
12/15/14	Ahmad Maruf	added sample.bc and imp.bc for project demo
12/15/14	wmcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/15/14	wmcu	import: bfs->dfs
12/15/14	Lingyuan He	fix bad cases ideal output
12/15/14	wmcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/15/14	wmcu	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	wmcu	fix test.sh
12/13/14	wmcu	fix test.sh
12/13/14	wmcu	fix test.sh
12/13/14	wmcu	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	wmcu	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	wmcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/13/14	wmcu	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	wmcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/13/14	wmcu	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	wmcu	fix built-in function empty
12/13/14	wmcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/13/14	wmcu	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	wmcu	error if call a function not implemented
12/13/14	wmcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/13/14	wmcu	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	wmcu	exit code fixed. 1 on failure
12/13/14	wmcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/13/14	wmcu	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	wmcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/12/14	wmcu	minor fix
12/12/14	Lingyuan He	matsub check in translate and fix bc lib
12/12/14	wmcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/12/14	wmcu	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	wmcu	update README
12/12/14	wmcu	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	wmcu	aa
12/12/14	wmcu	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	wmcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/12/14	wmcu	fix issue 55
12/12/14	Lingyuan He	dev_test makefile fix
12/12/14	wmcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/12/14	wmcu	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	wmcu	clean dev_test
12/12/14	wmcu	add runtime check to matsub
12/12/14	wmcu	fix string mat add
12/12/14	wmcu	fix issue 50
12/12/14	wmcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/12/14	wmcu	show filename in error message
12/12/14	wmcu	shoe filename in error message
12/12/14	wmcu	dev_test
12/12/14	wmcu	add dev_test
12/12/14	wmcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/12/14	wmcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/12/14	wmcu	Merge branch 'import_dev' into mat_dev
12/12/14	wmcu	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	wmcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/12/14	wmcu	fix ftbl
12/12/14	wmcu	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	wmcu	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	wmcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	wmcu	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	wmcu	Merge branch 'mat_dev' into import_dev
12/11/14	wmcu	remove some unsed variable
12/11/14	wmcu	isl fixed
12/11/14	Lan Yang	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	wmcu	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	wmcu	fix std::invalid_argument
12/11/14	Lan Yang	test
12/11/14	wmcu	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	wmcu	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	wmcu	merge
12/11/14	wmcu	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	wmcu	fix cols
12/11/14	wmcu	small formatting in translate.ml
12/11/14	wmcu	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	wmcu	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	wmcu	fix tid in CntFor
12/11/14	wmcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	Lan Yang	lib
12/11/14	wmcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/11/14	wmcu	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	wmcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/10/14	wmcu	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	wmcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/10/14	wmcu	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	wmcu	
12/10/14	wmcu	merge
12/10/14	wmcu	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	wmcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/10/14	wmcu	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	wmcu	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	wmcu	fix int-double match in function call
12/10/14	Ahmad Maruf	Merge branch 'master' of https://github.com/Maruf789/PLT
12/10/14	wmcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/10/14	wmcu	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	wmcu	formatting
12/10/14	wmcu	remove Failure
12/10/14	wmcu	formating
12/10/14	wmcu	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	wmcu	small bug
12/9/14	Lan Yang	test
12/9/14	wmcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/9/14	wmcu	fix (=)
12/9/14	Lan Yang	add two repositories
12/9/14	wmcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/9/14	wmcu	fix double quote in string
12/9/14	Lan Yang	merge
12/9/14	Lan Yang	sample2
12/9/14	wmcu	type equality: int = double
12/9/14	Lan Yang	merge
12/9/14	Lan Yang	merge

12/9/14	wmcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/9/14	wmcu	array code []
12/9/14	Lingyuan He	Disp
12/9/14	Lan Yang	merge
12/9/14	wmcu	CntFor v0.1
12/9/14	Lan Yang	merge
12/9/14	Lan Yang	merge
12/9/14	Lingyuan He	disp
12/9/14	wmcu	while end
12/9/14	wmcu	v0.2 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	wmcu	done
12/9/14	Ahmad Maruf	updated statement portion
12/9/14	wmcu	gen_stmt v0.5
12/9/14	wmcu	gen_stmt
12/9/14	wmcu	merge
12/9/14	wmcu	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	wmcu	Merge branch 'mat_dev' of https://github.com/Maruf789/PLT into mat_dev
12/9/14	wmcu	temp
12/9/14	Lan Yang	nothing
12/9/14	wmcu	rm ir.ml
12/9/14	wmcu	codegen v0.1
12/9/14	Ahmad Maruf	Deleted ir.ml
12/9/14	Lan Yang	test on translator
12/9/14	wmcu	translate v0.1
12/9/14	Lan Yang	modified test.sh
12/9/14	wmcu	nothing

12/9/14	wmcu	tmp
12/8/14	wmcu	merge
12/8/14	wmcu	fix issue #26
12/7/14	wmcu	test translate v0.1 - no mat support
12/7/14	wmcu	Merge branch 'master' of https://github.com/Maruf789/PLT
12/7/14	wmcu	first attempt on matval
12/7/14	Meng Wang	add column and row numbers to SMatval
12/5/14	wmcu	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	wmcu	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	wmcu	fix parser issues
11/30/14	wmcu	trans_expr v0.1
11/30/14	Lan Yang	modified test cases
11/30/14	wmcu	smart test
11/30/14	wmcu	make clean
11/30/14	wmcu	fix parser: function declare
11/30/14	wmcu	Merge branch 'test_scanner_parser' of https://github.com/Maruf789/PLT into test_scanner_parser
11/30/14	wmcu	scheck done
11/29/14	Lan Yang	most basic test cases done
11/29/14	Lan Yang	more test cases
11/29/14	wmcu	add library function
11/28/14	wmcu	function: take arguments into variable table
11/28/14	wmcu	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	wmcu	fix tab
11/23/14	wmcu	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	wmcu	add \n to perror
11/23/14	wmcu	print error message to stderr now
11/23/14	wmcu	add comments in main
11/23/14	wmcu	Merge branch 'test_scanner_parser' of https://github.com/Maruf789/PLT into test_scanner_parser
11/23/14	wmcu	more precise error message
11/23/14	Lan Yang	test
11/23/14	wmcu	Add Scanner_error

11/23/14	wmcu	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	wmcu	check_sexpr v0.3
11/22/14	wmcu	scheck_expr v0.3
11/22/14	wmcu	scheck_expr v0.2
11/19/14	wmcu	ignore main.bin
11/19/14	wmcu	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 sample0 to R
11/15/14	wmcu	Add a README in src
11/14/14	wmcu	remove unused files
11/14/14	wmcu	can compile hello world
11/14/14	wmcu	split scheck.ml and translate.ml
11/13/14	wmcu	return type of expr
11/12/14	wmcu	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	wmcu	fix bug in string
11/10/14	wmcu	fix sample0
11/10/14	wmcu	translate v0.5
11/10/14	wmcu	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