Facelab

FACELAB: A PROGRAMMING LANGUAGE TO MANIPULATE PORTRAIT PHOTOS

Group members:	*UNI:*
Xin Chen	xc2409
Kejia Chen	kc3136
Tongfei Guo	tg2616
Weiman Sun	ws2517

Contents

1 Introduction **5**

1.1 Context . . . 5

1.2 Aims and Motivations . . . 5

2 Tutorial **6**

2.1 Environment Setup . . . 6

2.2 Using the Compiler . . . 7

2.3 Sample Program . . . 7

2.3.1 Hello World . . . 7

3 Language Reference Manual **8**

3.1 Types . . . 8

3.1.1 Basic data types . . . 8

3.2 Lexical Conventions . . . 8

3.2.1 Identifiers . . . 8

3.2.2 keywords . . . 8

3.2.3 Literals . . . 9

3.2.4 Comments . . . 10

3.2.5 Operators . . . 10

3.2.6 punctuator . . . 11

3.3 Syntax Notations . . . 11

3.3.1 Expressions . . . 11

3.3.2 Primary Expressions . . . 12

3.3.3 Postfix Expressions . . . 12

3.3.4 Subscripts . . . 12

3.3.5 Function Calls . . . 12

3.4 Declarations . . . 12

3.4.1 Type specifiers . . . 12

3.4.2 Matrix declarations . . . 13

3.4.3 Function declarations . . . 13

3.5 Standard Libraries Functions . . . 13

3.5.1 image-related . . . 13

3.5.2 output . . . 14

3.6 Rules and Sample Programs . . . 14

3.6.1 Variable Declaration . . . 14

3.6.2 Invoking functions and multiple returns . . . 16

3.6.3 Scoping . . . 16

3.6.4 GCD Algorithm . . . 17

3.6.5 Apply a Filter . . . 18

3.6.6 Face Detection 18
3.6.7 photo editing 20
3.7 Built-in Functions 21

4 Architecture **22**
4.1 Diagram 22
4.2 Compiler 24

5 Project Plan **26**
5.1 Timeline 26
5.2 Team Roles 26

6 Test **29**
6.1 Test Cases 29

7 Lessons Learned **29**

8 Appendix **31**
8.1 preprocess 31
8.2 scanner 31
8.3 parser 33
8.4 AST 37
8.5 Semant 40
8.6 codegen 42
8.7 standard library 89
8.8 ext.cpp (opencv functions) 91
8.9 compile (shell script for calling Facelab compiler to generate .exe) 93
8.10 add1.fb 93
8.11 addDouble.fb 93
8.12 conv.fb 94
8.13 conv2.fb 95
8.14 double2int.fb 96
8.15 factorial.fb 96
8.16 gcd.fb 96
8.17 gcd_recursive.fb 97
8.18 int2double.fb 97
8.19 load_1.fb 98
8.20 load_2.fb 98
8.21 main_6.fb 98
8.22 main_7.fb 99
8.23 main_8.fb 99

8.24 main_9.fb . . . 99
8.25 matrix_1.fb . . . 100
8.26 matrix_2.fb . . . 100
8.27 matrix_3.fb . . . 100
8.28 matrix_4.fb . . . 101
8.29 matrix_5.fb . . . 101
8.30 matrix_6.fb . . . 102
8.31 matrix_7.fb . . . 102
8.32 matrix_9.fb . . . 102
8.33 matrix_11.fb . . . 103
8.34 matrix_13.fb . . . 103
8.35 matrix_14.fb . . . 103
8.36 matrix_15.fb . . . 104
8.37 multi_ret1.fb . . . 104
8.38 multi_ret2.fb . . . 104
8.39 printdouble.fb . . . 105
8.40 printdouble2.fb . . . 105
8.41 save_1.fb . . . 105
8.42 save_2.fb . . . 106
8.43 scope_1.fb . . . 106
8.44 scope_2.fb . . . 107
8.45 scope_3.fb . . . 107
8.46 scope_4.fb . . . 108
8.47 scope_5.fb . . . 108
8.48 semant_assign_1.fb . . . 108
8.49 semant_assign_2.fb . . . 108
8.50 semant_assign_3.fb . . . 109
8.51 semant_assign_4.fb . . . 109
8.52 semant_assign_5.fb . . . 109
8.53 semant_func_2.fb . . . 109
8.54 semant_func_3.fb . . . 110
8.55 semant_func_rename_1.fb . . . 110
8.56 semant_func_rename_2.fb . . . 110
8.57 semant_local_1.fb . . . 110
8.58 semant_matrix_1.fb . . . 110
8.59 semant_matrix_2.fb . . . 111
8.60 semant_predicate_1.fb . . . 111
8.61 semant_predicate_2.fb . . . 111
8.62 semant_predicate_3.fb . . . 111
8.63 semant_unop_1.fb . . . 111

8.64 plot.fb 112
8.65 face_1.fb 113
8.66 sharpen.fb 114

1 Introduction

1.1 Context

Profile photos editing is a crucial part in the broad category of photo editing. While photoshop and some other edge-cutting softwares do a pretty good job at photo editing, more of them still requires a large amount of manual labor. There exists few programming languages/softwares that enable picture manipulation, while allows a decent automation at the same time. Therefore, it is helpful to desgin a programming language that allow to batch manipulate pictures and more importantly photo portraits by users' own needs.

1.2 Aims and Motivations

Facelab aims to perform face detection, face recognition, filter applying and photo sticker adding among other features which enable the target users to manipulate their portrait photos with ease and accuracy.

The basic syntax of this language largely resembles that of C, excluding some of the irrelevant details such as inheritance, template, etc. With the inclusion of the matrix data type that is common to many scientific programming languages, it not only facilitates image processing related computation, but also grants users the ability to manipulate photo on a pixel scale and allows users the freedom to define and tailor their own filter to individuals' preference.

Moreover, by having OpenCV linked, it provides access to some of the state-of-art face detection and face recognition algorithms which grants the power of fast batch portrait editing.

A combination of these afore-mentioned features could considerably simplify real-life tasks such as adjusting photo brightness and contrast, batch-editing photos, auto-applying facial pixelization, and so on.

2 Tutorial

2.1 Environment Setup

Facelab was developed in Ocaml, before using Facelab to program, make sure that Ocaml is installed properly. To do this, follow the steps:

Install Homebrew

For easy installation, install Homebrew package manager.

```
$ /usr/bin/ruby -e "$(curl -fsSL
 https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Install Opam and Configure

OPAM is the OCaml Package Manager to install Ocaml packages and libraries.

```
$ brew install opam
$ opam init
```

Install LLVM

Take note of where brew places the LLVM executables. It will show you the path to them under the CAVEATS section of the post-install terminal output. Also take note of the LLVM version installed.

```
$ brew install llvm
```

Setup Opam Environment

```
$ eval 'opam config env'
```

Install Ocaml LLVM library

Make sure the LLVM version number you installed in this step matches the version number installed by Homebrew.

```
$ opam install llvm.5.0
```

After everything is installed properly, you should be able to use the Facelab compiler.

Install g++ or clang++

User needs to have a version of g++ or clang++ in order to convert the assembly that our compiler generates to executables.

Install openCV

User needs to have openCV (c++ version) libraries installed before running Facelab compiler, a link to how to install openCV (c++) is provided in README.

2.2 Using the Compiler

Please refer to README file in Facelab for instructions on running the compiler.

2.3 Sample Program

2.3.1 Hello World

Before diving into any complicated programs, let's get an idea of how to write the simple Hello World program in Facelab. Without the existence of main function, we just pass the string "Hello World" into the printf function. Simple, right?

```
// HelloWorld.fb

printf("Hello World");
```

As you don't need to declare the prototype of the main function, you can write your small programs right off the bat.

3 Language Reference Manual

3.1 Types

3.1.1 Basic data types

Table 1: basic data types

type name	description
int	32-bit signed integer
double	64-bit float-point number
bool	1-bit boolean variable
string	array of ASCII characters
matrix	data structure storing 2D-double matrix of arbitrary size

3.2 Lexical Conventions

3.2.1 Identifiers

Identifiers consists of one or more characters where the leading character is a lowercase letter followed by a sequence uppercase/lowercase letters, digits and possibly underscores. Identifiers are primarily used variable declaration.

3.2.2 keywords

The keywords listed below are reversed by the language and therefore will not be able to be used for any other purposes (e.g. identifiers)

Table 2: keywords

type name	description
for	typical for loop follows the syntax *for(init; cond; incr) stat;*
while	typical while loop follows the syntax *while(cond) stat;*
if	typical if-elseif-else condition clause follows the syntax
elseif	*if(cond) stat; elseif(cond) stat; else stat;*
else	
return	ending current function execution and return a value or multiples values
func	signal word for function definition follow the syntax *func name(type var, ...) stat;*
true	boolean type constant
false	boolean type constant
int	32-bit signed integer
double	64-bit float-point number
bool	8-bit boolean variable
string	array of ASCII characters
matrix	data structure storing bool/ints/doubles of arbitrary size
save	build-in function name
load	build-in function name
face	build-in function name
filter	build-in function name

3.2.3 Literals

integer literals

A sequence of one or more digits representing an un-named(not associated with any identifier) integer, with the leading digit being non-zero (i.e. [1-9][0-9]*)

double literals

A sequence of digits seperated by a '.' representing an un-named float-point number (i.e. [0-9]*.[0-9]*)

matrix literals

A sequence of digits enclosed by a pair of square brackets, and delimited by commas and semi-colons, representing an un-named 2-D matrix.

e.g. [1.1, 2.2; 3.3, 4.4] = $\begin{bmatrix} 1.1 & 2.2 \\ 3.3 & 4.4 \end{bmatrix}$

string literals

A sequence of character enclosed by a pair of double quotation marks representing an un-named string. (i.e. ˆ".*" $)

3.2.4 Comments

Table 3: comments

/* comment */	block comment where comment could contain newline
// comment	line comment without newline

3.2.5 Operators

basic operators

Table 4: scalar operators

=	assignment operator
+, -, *, /	arithmetic operators
%	reminder operator
! =, ==, >, >=, <, <=	relational operators
||, &&, !	logical operators(OR, AND, NOT)

matrix operators

Table 5: matrix operators

=	assignment operator
+, -, *, /	arithmeitc operators for matrix
.*	matrix dot product
$M[i, j]$	subscript operator
$M[:, j]$	subscript j-th column
$M[i, :]$	subscript i-th row
$M[: i, : j]$	block indexing from row 0 to row i, col 0 to col j
$M[i :, j]$	block index from row i to the last row
$M[i\_low, i\_high, j\_low : j\_high]$	
block indexing $	pre-defined operator whose syntax follows $matrix \$ filter$, which ap

3.2.6 punctuator

Semicolons at the end of each statement perform no operation but signal the end of a statement. Statements must be separated by semicolons.

3.3 Syntax Notations

3.3.1 Expressions

Precedence and Associativity Rules

Table 6: Operator Precedence and Associativity

Tokens (From High to Low Priority)	Associativity
!	R-L
$	L-R
* / % .*	L-R
+ -	L-R
$< <= > >=$	L-R
== !=	L-R
&&	L-R
||	L-R

3.3.2 Primary Expressions

Identifiers, literals and parenthesized expressions are all considered as ”primary expressions”.

3.3.3 Postfix Expressions

Postfix expressions involving subscripting and function calls associate left to right. The syntax for these expressions is as follows:

postfix-expression:	primary-expression
	postfix-expression [expression]
	postfix-expression (argument-expression-list)
argument-expression-list:	argument-expression
	argument-expression-list, argument-expression

3.3.4 Subscripts

A postfix expression followed by an expression in square brackets is a subscript. For our 2-D matrix, the expression would be two values separated by a comma, the value could be an integer or a colon.

3.3.5 Function Calls

A function call is a postfix expression followed by parentheses containing a (possibly empty) comma-separated list of expressions that are the arguments to the function.

3.4 Declarations

3.4.1 Type specifiers

`int`
`double`
`bool`
`string`
`matrix`

Each variable declaration must be preceded by a type specifier which tells what type is going to be used to store that variable.

3.4.2 Matrix declarations

```
example:
matrix name = [a,b,c;d,e,f;g,h,i];
```

The **`matrix`** specifier define the variable as a matrix type. In the example, a–i are of type double. The value is surrounded by a pair of brackets. semi-colons are to separate different rows, where in every row, elements are separated by commas.

3.4.3 Function declarations

```
example:
func funcName(T arg1, ...)  {...}
```

To define a function, use the keyword **`func`** to declare this is a function declaration. Following by user defined function's name. In the parentheses it defines how many arguments it can be passed in and what types are they. Therefore in the calling environment, the calling statement must match the function's definition.

3.5 Standard Libraries Functions

3.5.1 image-related

Table 7: Standard Libraries Functions for image

Functions	Description
func load(string filename)	load image from a file
func save(matrix r, matrix g, matrix b, string filename)	save image to given filename
func face(string filename)	detect whether a image includes faces

3.5.2 output

Table 8: Standard Libraries Functions for I/O

Functions	Description
func printf(string str)	print a string
func printf(matrix m)	print a matrix
func printf(int i)	print integer i
func printf(double d)	print double d
func printf(bool b)	print bool ”true” or ”false”
func printend()	print a new line, and the next printing statement will automatically start with a new line

3.6 Rules and Sample Programs

In general, every statement must end with a semicolon ";". Code blocks in control flow statements (if, else, elseif, for, while) must always be enclosed in braces. Braces can also form blocks in non-flow-control statements, and each block forms its own new scope (with static scoping rule). The program begins from top down, statements can be interleaved with function definitions, both function names and variable names follow normal static scoping rule. functions definition don't have return types in the prototype, but can return any type and any number of variables in the function. Every function has an argument that takes 0 or more variables, surrounded by parentheses. When calling a function, the number of variables passed into the calling function must match its arguments and corresponding types. If a return object from a function is being stored in a variable, the variable type must match the type of the return object from the function, and if the function returns multiple values, then both the types and the number of variables that's being assigned to must match accordingly.

3.6.1 Variable Declaration

string

A string in Facelab is defined as string literal, surrounding by a pair of double quotation marks.

```
string s1 = "My string";
string s2;
s2 = "This is another string";
```

```
printf(s2);//this will output "This is another string"
```

int, double

The int data type is a 32-bit signed two's complement integer, which has a minimum value of -2^31 and a maximum value of 2^31-1.
The double data type is a double-precision 64-bit.
There are **int2double** and **double2int** built-in functions to convert the values between the two types.

```
double d1 = 0.0;
double d2 = 1.111;
double sum;
sum = d1 + d2; // sum == 1.111

int num = 1;
printf(num == int2double(sum)); // this will output true;
```

matrix

Matrix has its own operations. Before doing any operation, the dimension of each operation mush agree.
i). Between a scalar and a matrix : matrix op number | number op matrix (op : + - * /)
ii). Between two matrices : matrix op matrix (op : + - * / $)
iii). matrix dot product : matrix .* matrix
iv). matrix indexing : Syntax-wise resembles Matlab matrix indexing rules.
matrix[x1, y1] | matrix[x1:x2, y1:y2] | matrix[x1:, y1] | matrix[:, y1] | matrix[:, :y2] | etc.

```
matrix m1 = [3.1, 3.0; 2.1, 2.0; 1.1, 1.0]; // 3 by 2 matrix
matrix m2 = [0.0, 0.1, 0.2; 1.0, 1.1, 1.2]; // 2 by 3 matrix

matrix m3;
m3 = m1 .* m2;
printf(m3);
// m3 is the dot product of m1 and m2, resulting a 3 by 3 matrix
//3.000000 3.610000 4.220000
//2.000000 2.410000 2.820000
//1.000000 1.210000 1.420000
```

```
printf(m3[1:,2]); // this prints out a submatrix of m3
                  //2.820000
                  //1.420000
printf(m2[0,0] == 0.0); // this prints out true, since
    m1[0,0]=0.0 and 1 by 1 matrix is also viewed as a single
    number
```

3.6.2 Invoking functions and multiple returns

Define a function before calling it. The passing variables should match the number of variables and the corresponding types in the functions argument. You can return multiple variables and they don't have to be the same type.

```
func myFunction(int a, double b, matrix c){
   a = a + 1;
    return a, c[0,0]==b;
}
int a = 5;
bool foo;
a, foo = myFunction(a, 2.3, [1.5,9.3]);
printf(a);
printend();
printf(foo);
/* the program will printout:
6
false
*/
```

3.6.3 Scoping

Facelab utilizes static scoping, which means if a variable is created in a pair of curly brackets, it can't be seen out of the bracket.
For example:

```
int i = 0;
{
    int j = 5;
    printf(i);
    printf(j);
```

```
    {
        i = 1;
        int j = 6;
        printf(i);
        printf(j);
        {
            i = 2; // this is still the i in the first line
        }
    }
    {
        {
             int i;
             i = 3; // value 3 is not visible after this curly
                 bracket
        }
    }
}
printf(i); // print out 2
//printf(j);// this will give error: variable j not declared.
```

3.6.4 GCD Algorithm

```
func gcd(int m, int n) {
//calculate gcd of two integer number
    while(m>0)
    {
        int c = n % m;
        n = m;
        m = c;
    }
    return n;
}

func gcd_recursive(int m, int n)
{
    if (m == 0)
        return n;
    if (n == 0)
        return m;
    if (m > n)
        return gcd(m%n, n);
```

```
   else
      return gcd(n%m, m);
}
```

3.6.5 Apply a Filter

```
matrix t_r; matrix t_g; matrix t_b;
t_r,t_g,t_b = load("sbird.jpg");
matrix r_r; matrix r_g; matrix r_b;
matrix r2_r; matrix r2_g; matrix r2_b;

matrix s = [0.0, -1.0, 0.0;
        -1.0, 5.0, -1.0;
         0.0, -1.0, 0.0]; //sharpen filter

matrix s2 = [1.0, 4.0, 6.0,4.0,1.0;
         4.0, 16.0,24.0, 16.0,4.0;
         6.0, 24.0, 36.0, 24.0,6.0;
         4.0, 16.0,24.0, 16.0,4.0;
         1.0, 4.0, 6.0,4.0,1.0] / 35.0; //Gaussian blur and
            eliminate background
r_r = t_r $ s;
r_g = t_g $ s;
r_b = t_b $ s;
save(r_r, r_g, r_b, "sbird_result.jpg");
r2_r = t_r $ s $ s2;
r2_g = t_g $ s $ s2;
r2_b = t_b $ s $ s2;
save(r2_r, r2_g, r2_b, "sbird_result2.jpg");
```

3.6.6 Face Detection

```
matrix m;
m = face("b.jpg");
matrix m_r; matrix m_g; matrix m_b;
m_r, m_g, m_b = load("b.jpg");
double x = m[0,0]; double y = m[1,0]; double l = m[2,0]; double
   w = m[3,0];
int i;
```

Figure 1: original

Figure 2: apply filter s

Figure 3: apply filter s and s2

```
for (i = double2int(x - l/2); i <= double2int(x +l/2); i = i+1)
{
   m_g[i, double2int(y-w/2-2):double2int(y-w/2+2)] =
      (255.0-zeros(1,5));
   m_b[i, double2int(y-w/2-2):double2int(y-w/2+2)] =
      (255.0-zeros(1,5));
   m_r[i, double2int(y-w/2-2):double2int(y-w/2+2)] = zeros(1,5);
   m_g[i, double2int(y+w/2-2):double2int(y+w/2+2)] =
      (255.0-zeros(1,5));
   m_b[i, double2int(y+w/2-2):double2int(y+w/2+2)] =
      (255.0-zeros(1,5));
   m_r[i, double2int(y+w/2-2):double2int(y+w/2+2)] = zeros(1,5);
}
for (i = double2int(y - w/2); i <= double2int(y +w/2); i = i+1)
{
   m_g[double2int(x-l/2-2):double2int(x-l/2+2), i] =
      (255.0-zeros(5,1));
   m_b[double2int(x-l/2-2):double2int(x-l/2+2), i] =
      (255.0-zeros(5,1));
   m_r[double2int(x-l/2-2):double2int(x-l/2+2), i] = zeros(5,1);
   m_g[double2int(x+l/2-2):double2int(x+l/2+2), i] =
      (255.0-zeros(5,1));
   m_b[double2int(x+l/2-2):double2int(x+l/2+2), i] =
      (255.0-zeros(5,1));
   m_r[double2int(x+l/2-2):double2int(x+l/2+2), i] = zeros(5,1);
}
save(m_r, m_g, m_b, "face_2_result.jpg");
```

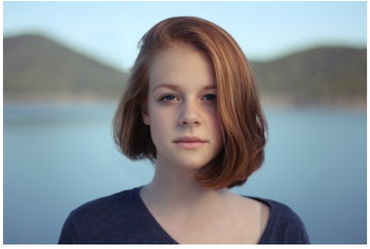


Figure 4: original

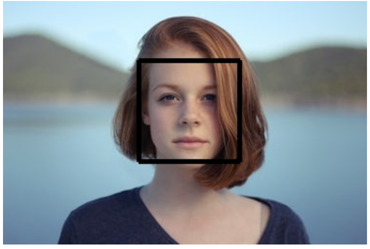


Figure 5: after face detection

3.6.7 photo editing

```
matrix t_r; matrix t_g; matrix t_b;
t_r,t_g,t_b = load("tshirt.jpg");
matrix e_r; matrix e_g; matrix e_b;
e_r,e_g,e_b = load("edwards.jpg");
int row_t; int col_t; int row_e; int col_e;
row_t, col_t = size(t_r);
row_e, col_e = size(e_r);
matrix m;
m = face("edwards.jpg");
int start_x=double2int(m[0,0]+m[2,0]/2+1); int
   start_y=double2int(m[1,0]-col_t/2+1);
int i; int j;
for (i = 0; i!= row_t; i=i+1)
{
   for (j=0; j!=col_t; j=j+1)
   {
      if ((t_r[i,j] <= 252.0) && (t_g[i,j] <= 252.0) &&
          (t_b[i,j] <= 252.0))
      {
         if ((start_x+i < row_e) && (start_y+j < col_e))
         {
            e_r[start_x+i,start_y+j] = t_r[i,j];
            e_g[start_x+i,start_y+j] = t_g[i,j];
            e_b[start_x+i,start_y+j] = t_b[i,j];
         }
      }
   }
}
save(e_r, e_g, e_b, "nerd_edwards.jpg");
```

Figure 6: original 1

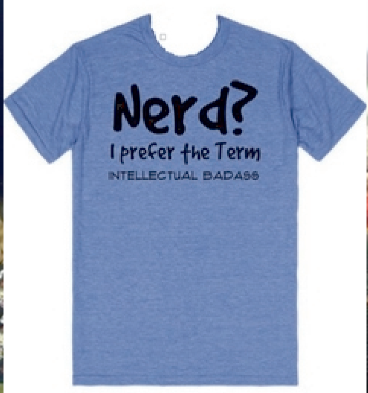


Figure 7: orignial 2

Figure 8: result

3.7 Built-in Functions

size

size function takes a matrix as argument, and returns the size of a matrix by a pair of int, which indicate the number of rows and columns.

```
i, j = size(m)
```

zeros

zeros takes two int as arguments, indicating row and column numbers, and returns a matrix will all zero entries with the designated size.

```
m = zeros(i, j)
```

int2double

int2double takes an int as argument and returns a double type with that value.

```
d = int2double(i)
```

double2int

double2int takes a double as argument and cast into an int. Decimal points will be truncated.

```
d = double2int(i)
```

save

save takes three matrices representing red, green, blue as its RGB values, and a path string as arguments. So to save the image to path.

```
save(m_r, m_g, m_b, path)
```

load

load takes a path string of an image as argument, and returns three matrices representing red, green, blue as its RGB values.

```
m_r, m_g, m_b = load(path)
```

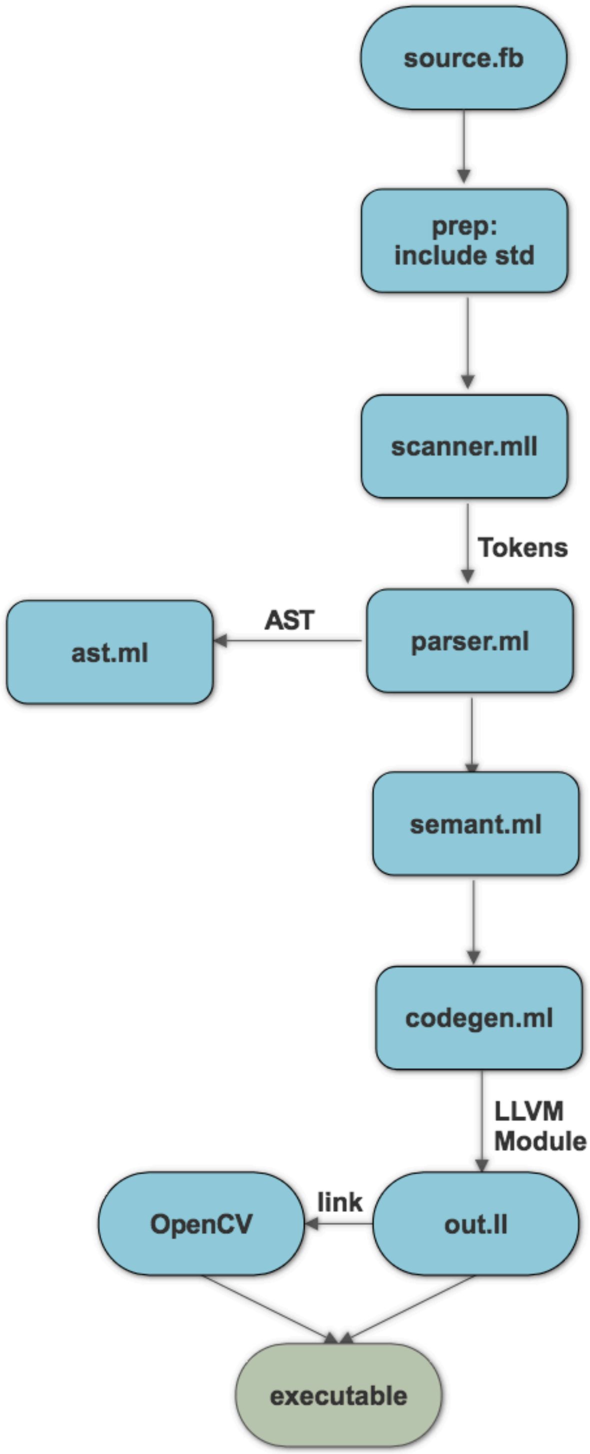
face

Detect faces in the image at given path, return m is a 4 by n matrix, n is the number of faces, row 1 stores coordinates of the center of faces at which row, row 2 stores coordinates of the center of faces at which col, row 3 stores the height of the faces, row 4 stores the length of faces.

```
m = face(path)
```

4 Architecture

4.1 Diagram


source.fb
prep:
include std
scanner.mll
Tokens
AST
ast.ml
parser.ml
semant.ml
codegen.ml
LLVM
Module
link
OpenCV
out.ll
executable

4.2 Compiler

facelab.ml (Top level)

This is the top-level of Facelab compiler, it invokes the prep, scanner, parser, semant, and codegen modules to generate the LLVM IR, and dumps the module.

source.fb

This is the top level Facelab program that needs to be compiled.

prep.ml

Include any standard libraries.

scanner.mll

After the preparation of the source file, scanner reads the source Facelab code and does the lexical analysis. Tokenizing codes from the input source code. If there is illegal character then the lexicon would not pass. If passed, then the tokens are passed to the parser.

parser.mly

Read tokens from scanner module, make sure they are syntactically correct. If the process of parsing has no error occurred, it will generate the abstract syntax tree(AST).

ast

The abstract syntax tree representation of the Facelab program.

semant.ml

It is the checker to make sure AST is semantically correct. It takes in the AST representation and, if all checks are passed, pass the AST representation to the codegen module.

codegen.ml

After the semant of AST was checked, codegen takes in AST and convert it into an out file. It's worthnoting that many of the semantic checks are done in the stage during the necessity of run-time error checking. For instance, if a matrix subscript is a non-literal expression, and sometimes it's just more convenient to check it here. The out file is an LLVM bytecode.

OpenCV

OpenCV is linked with the LLVM bytecode to produce assembly code of executable. It provides load, save functionality in our case, and a face detection function.

5 Project Plan

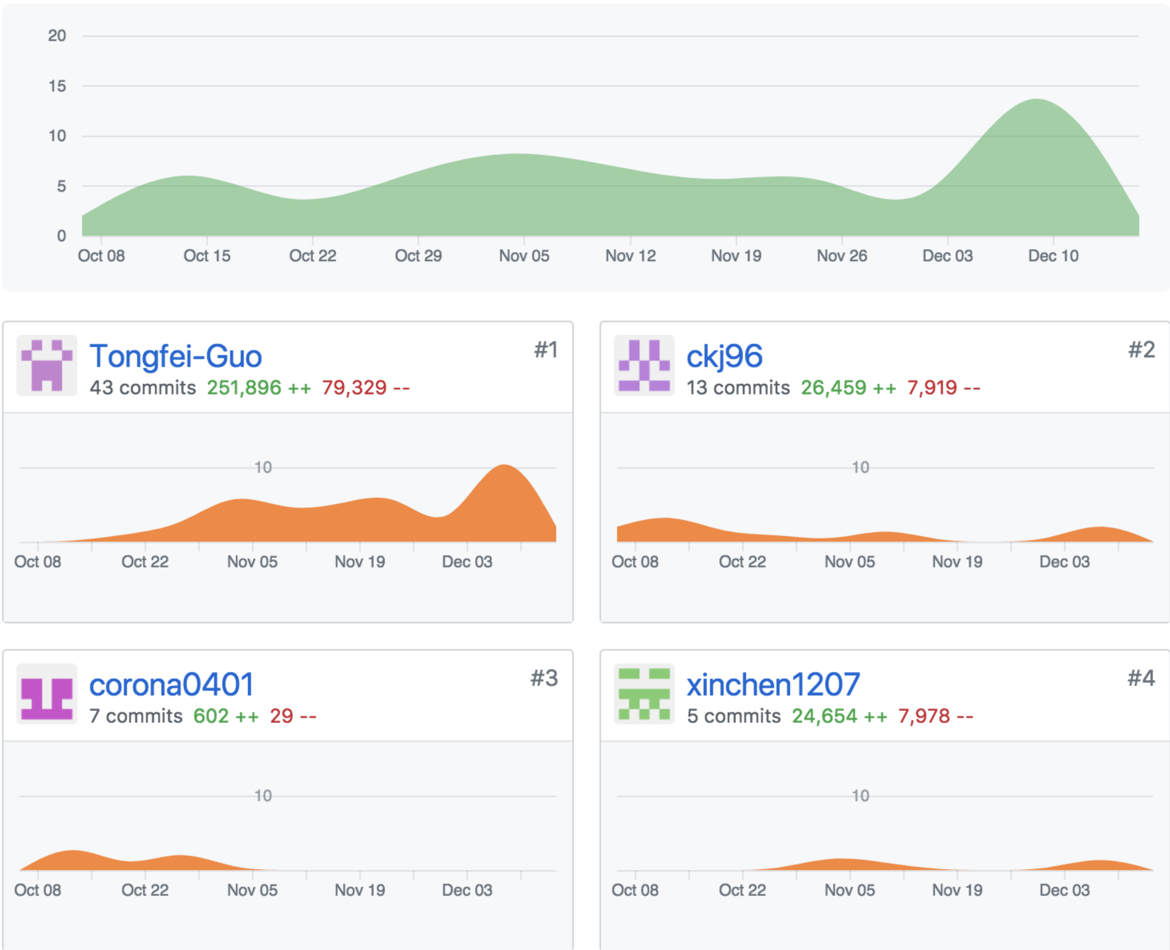
5.1 Timeline

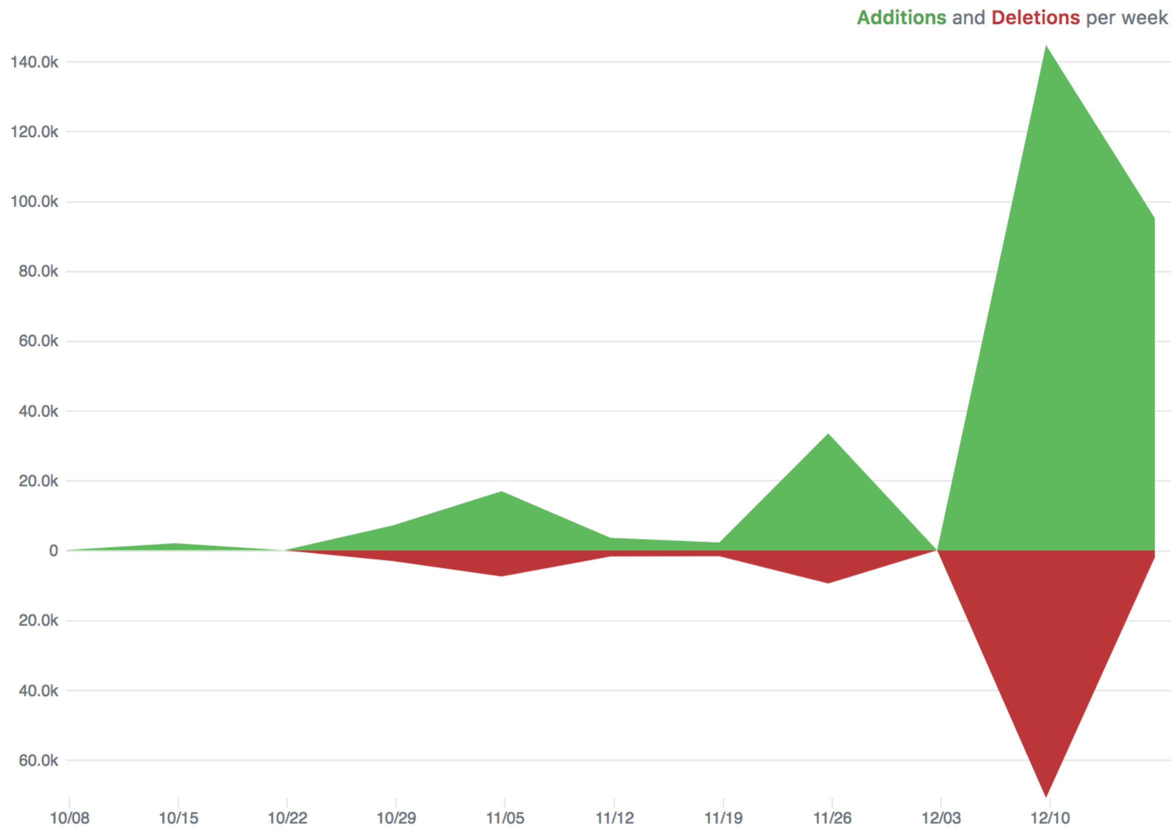
Table 9: Timeline table

Date	Accomplishment
Sep 20	First group meeting, decided what kind of language we want to design. Start working on the project proposal composing
Sep 25	Submitted the project proposal, got the feedback from TA therefore officially determined to implement Facelab programming language.
Oct 4	Worked on scanner, parser and AST. Clarify all the syntax and rules.
Oct 16	Finished the Language Reference Manual.
Nov 5	Implemented built-in function printf to make sure 'Hello World' works properly.
Nov 8	Enabled statements without main()
Nov 15	Added matrix data type and the matrix-wise operations. Also slicing matrix into sub matrices is enabled
Nov 20	Finished matrix auxiliaries
Nov 28	Redid type inference and enabled multiple return values
Dec 14	Added sematic check.
Dec 15	Filter was enabled.
Dec 16	Load and Save functions were added.
Dec 17	Successfully linked to OpenCV to utilize its face recognition functions, added some built-in functions.

5.2 Team Roles

Member	Role	Work Done
Weiman Sun	Manager	scanner, parser, load&save, OpenCV, testing
Tongfei Guo	Language Guru	design syntax, scanner, parser, ast, codegen, testing
Kejia Chen	System Architect	scanner, parser, semant, codegen, preprocess, filter
Xin Chen	Tester	scanner, parser, testing, final report composing


20
15
10
5
0
Oct 08
Oct 15
Oct 22
Oct 29
Nov 05
Nov 12
Nov 19
Nov 26
Dec 03
Dec 10
Tongfei-Guo
#1
43 commits 251,896 ++ 79,329 --
10
Oct 08
Oct 22
Nov 05
Nov 19
Dec 03
ckj96
#2
13 commits 26,459 ++ 7,919 --
10
Oct 08
Oct 22
Nov 05
Nov 19
Dec 03
corona0401
#3
7 commits 602 ++ 29 --
10
Oct 08
Oct 22
Nov 05
Nov 19
Dec 03
xinchen1207
#4
5 commits 24,654 ++ 7,978 --
10
Oct 08
Oct 22
Nov 05
Nov 19
Dec 03


Additions and Deletions per week
140.0k
120.0k
100.0k
80.0k
60.0k
40.0k
20.0k
0
20.0k
40.0k
60.0k
10/08
10/15
10/22
10/29
11/05
11/12
11/19
11/26
12/03
12/10

6 Test

6.1 Test Cases

Test cases are written to test the correctness of syntax, semantics and functions. Each test case was targeting on one feature as the developing phase. They could be written by anyone who intended to test his own implementation. If a case was failed, it was marked so that later the developer could notice where to improve the quality. You can view all test cases code either at the appendix of the report or refer to Facelab.tar.gz

7 Lessons Learned

Kejia Chen

I believe I learnt quite a lot from the course and project this semester. The design of the language at the beginning is a even more challenging part than implementation I think. We choose a C-like design with low risk but I wish we could think of another kind of syntax to create something innovative. The project sounds scary for one who does not know much about language and compiler. But it turns out to be quite smooth thanks to my teammates. Since we decide to implement a C-like language, it's actually not that hard at the beginning with micro c compiler provided by Prof.Edwards. However, when it comes to the middle of the semester, we are really confused on how we can add our features like matrix or external library. My suggestion is that do not try to add all features at once, it will be harder to debug and test. Instead, you should make your complier runnable every time you add a new feature. Overall, it's a truly challenging yet rewarding one. that feeling is amazing when your compiler finally works as expected and do something you even cannot think of when you design it.

Xin Chen

This is the first time I have learned the programing languages in a compiler level, so it was a lot to take in. Implementing a new language in Ocaml was very challenging as the syntax of Ocaml could be convoluted and intimidating. The key to this class is to start early, otherwise there will be insane workload to work on down the road. Communication and being supportive to your team is so important since everyone has his own strengths and weaknesses, taking the responsibilities would make the teamwork much more efficient.

Suggestions: If time allows, learning the basic syntax of Ocaml beforehand since that will save the time in the beginning of semester and allow you to start working on your compiler right away. During the process of your project, there will be difficult obstacles, However Edward and TAs are there to help, do not leave your unsolved questions until the last minute when everything is too late.

Tongfei Guo

It's quite some fun, learning and more importantly implementing a compiler from scratch, though it's pity that it stops at IR without getting deep into optimization and other lower-level stuff. In case some future students refer to this report, a word of advice, include as much information as possible in your AST, anything you think might be useful. Storing a redundant AST is not that expensive, but if you later realize that you actually need something from AST which you did not store, it would be much of a hassle to add it in. This is why I had to check matrix size at run-time instead of compile-time.

Weiman Sun

I've never done such a big project before. Reading OCaml is a pain for me at first, but when I get it, everything becomes so clear and I definitely realize I can make a grete difference with so little code. Thanks for my teammates' hard effort, it was my pleasure to work with them. Suggetions: find a good team and start early.

8 Appendix

8.1 preprocess

```
let process_file filename1 =
  let read_all_lines file_name =
   let lines = ref [] in
    let chan = open_in file_name in
    try
      while true; do
        lines := input_line chan :: !lines
      done; []
    with End_of_file ->
      close_in chan;
    List.rev !lines
  in
  let concat = List.fold_left (fun a x -> a ^ x) "" in
  " \n " ^ concat (read_all_lines filename1) ^ " \n "
```

8.2 scanner

```
(* Ocamllex scanner for Facelab *)

{ open Parser }

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }     (* Comments *)
| "//"     { quote lexbuf}
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| '['      { LBRACKET }
| ']'      { RBRACKET }
| ';'      { SEMI }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
```

```
| '/'    { DIVIDE }
| '%'    { REMAINDER }
| '='    { ASSIGN }
| '$'    { FILTER }
| ':'    { COLON }
| ".*"   { MATPRODUCT }
| "=="   { EQ }
| "!="   { NEQ }
| '<'    { LT }
| "<="   { LEQ }
| ">"    { GT }
| ">="   { GEQ }
| "&&"   { AND }
| "||"   { OR }
| "!"    { NOT }
| "if"   { IF }
| "else" { ELSE }
| "elseif" { ELIF }
| "for"  { FOR }
| "while" { WHILE }
| "return" { RETURN }
| "break" { BREAK }
| "continue" { CONTINUE }
| "func" { FUNCTION }
| "matrix" { MATRIX }
| "image" { IMAGE }
| "int"  { INT }
| "double" { DOUBLE }
| "string" { STRING }
| "bool" { BOOL }
| "void" { VOID }
| "true" { TRUE }
| "false" { FALSE }
| ['0'-'9']+ as lxm { INT_LITERAL(int_of_string lxm) }
| ['0'-'9']+'.'['0'-'9']+ as lxm {
  DOUBLE_LITERAL(float_of_string lxm)}
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm {
  ID(lxm) }
| '"' ([^ '"']* as lxm) '"' { STRING_LITERAL(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped
  char)) }
```

```
and comment = parse
  "*/" { token lexbuf }
| _   { comment lexbuf }

and quote = parse
  ['\n' '\r'] { token lexbuf }
| _   { quote lexbuf }
```

8.3 parser

```
/* Ocamlyacc parser for MicroC */

%{
open Ast
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE LBRACKET RBRACKET COLON
    COMMA ID_SEP_COMMA
%token PLUS MINUS TIMES DIVIDE ASSIGN NOT REMAINDER MATPRODUCT
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
%token RETURN IF ELSE FOR WHILE INT DOUBLE BOOL STRING ELIF
    BREAK CONTINUE VOID
%token FUNCTION MATRIX IMAGE
%token FILTER
%token <int> INT_LITERAL
%token <float> DOUBLE_LITERAL
%token <string> STRING_LITERAL
%token <string> ID
%token GLOBAL EOF


%left SEMI
%nonassoc RETURN
%right ASSIGN
%nonassoc NOELSE
%nonassoc ELSE
%nonassoc ELSEIF
%left COMMA
%nonassoc COLON
%left OR
%left AND
```

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

%start program
%type <Ast.program> program

%%

program:
  decls EOF { let (fst, snd) = $1 in (List.rev fst, List.rev
      snd) }

decls:
   /* nothing */ { [], [] }
 | decls fdecl { ($2 :: fst $1), snd $1 }
 | decls stmt { fst $1, ($2 :: snd $1) }

fdecl:
   FUNCTION ID LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE
     { { typ = Void;
      fname = $2;
    formals = $4;
    body = List.rev $7 } }

formals_opt:
    /* nothing */ { [] }
  | formal_list { List.rev $1 }

formal_list:
    typ ID                { [($1,$2)] }
  | formal_list COMMA typ ID { ($3,$4) :: $1 }

typ:
    INT { Int }
  | DOUBLE { Double }
  | BOOL { Bool }
  | VOID { Void}
  | IMAGE {Image}
  | MATRIX {Matrix}
```

```
  | STRING {String}

stmt_list:
    /* nothing */ { [] }
  | stmt_list stmt { $2 :: $1 }

stmt:
    expr SEMI { Expr $1 }
  | RETURN SEMI { Return Noexpr }
  | RETURN expr SEMI { Return $2 }
  | LBRACE stmt_list RBRACE { Block(List.rev $2) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5,
     Block([])) }
  /* elseif */
  | IF LPAREN expr RPAREN stmt ELSE stmt { If($3, $5, $7) }
  | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
     { For($3, $5, $7, $9) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5) }
  | typ ID SEMI { Local($1, $2, Noassign) }
  | typ ID ASSIGN expr SEMI { Local($1, $2, $4) }


expr_opt:
    /* nothing */ { Noexpr }
  | expr       { $1 }

expr:
    INT_LITERAL  { IntLit($1) }
  | STRING_LITERAL { StringLit($1) }
  | DOUBLE_LITERAL { DoubleLit($1) }
  | double_mat_literal { MatrixLit(fst $1, snd $1) }
  | TRUE          { BoolLit(true) }
  | FALSE         { BoolLit(false) }
  | ID            { Id($1) }
  | expr PLUS expr { Binop($1, Add, $3) }
  | expr MINUS expr { Binop($1, Sub, $3) }
  | expr TIMES expr { Binop($1, Mult, $3) }
  | expr DIVIDE expr { Binop($1, Div, $3) }
  | expr MATPRODUCT expr { Binop($1, Matprod, $3) }
  | expr FILTER expr { Binop($1, Filter, $3) }
  | expr REMAINDER expr{ Binop($1, Rmdr, $3) }
  | expr EQ  expr { Binop($1, Equal, $3) }
  | expr NEQ expr { Binop($1, Neq, $3) }
```

```
  | expr LT  expr { Binop($1, Less, $3) }
  | expr LEQ expr { Binop($1, Leq, $3) }
  | expr GT  expr { Binop($1, Greater, $3) }
  | expr GEQ expr { Binop($1, Geq, $3) }
  | expr AND expr { Binop($1, And, $3) }
  | expr OR  expr { Binop($1, Or, $3) }
  | expr COMMA expr { match $1, $3 with
                        Comma(e1), Comma(e2) -> Comma(e1@e2)
                      | Comma(e1), e2 -> Comma(e1@[e2])
                      | e1, Comma(e2) -> Comma(e1::e2)
                      | e1, e2 -> Comma([e1;e2])
                    } /* a lot of sematic check needs for this one,
                        the only cases it's allow is in return expr,
                        ID LPAREN expr_opt RPAREN, and expr ASSIGN
                        expr*/
  | MINUS expr %prec NEG { Unop(Neg, $2) }
  | NOT expr     { Unop(Not, $2) }
  | expr ASSIGN expr { Assign($1, $3) } /*add to semant, check
     here only id and matrix indexing can be assigned to, left
     hand side can be multiple left value, right hand side can
     be not be expr COMMA expr */
  | ID LBRACKET expr RBRACKET { match $3 with
                                  Comma([e1;e2]) ->
                                    let r1 =
                                      (match e1 with
                                        Range(_,_) -> e1
                                      | _ -> Range(ExprInd(e1),
                                          ExprInd(e1)))
                                    and r2 =
                                      (match e2 with
                                        Range(_,_) -> e2
                                      | _ -> Range(ExprInd(e2),
                                          ExprInd(e2)))
                                    in
                                    Index($1, (r1,r2))
                                | _ -> failwith("wrong indexing
                                    expression")
                              }
  | ID LPAREN expr_opt RPAREN { let actuals =
                                  match $3 with
                                    Comma e1 -> e1
                                  | Noexpr -> []
                                  | _ -> [$3]
```

```
                              in
                              Call($1, actuals) }
  | LPAREN expr RPAREN { $2 }
  /* expr below are for matrix indexing only */
  | expr COLON       { Range(ExprInd($1), End) }
  | expr COLON expr  { Range(ExprInd($1), ExprInd($3)) }
  | COLON expr       { Range(Beg, ExprInd($2)) }
  | COLON                  { Range(Beg, End) }


double_mat_literal: /* matrix parsing */
    LBRACKET RBRACKET { [|[| |]|], (0, 0) } /* empty matrix */
  | LBRACKET double_mat_rows RBRACKET { $2 }

double_mat_rows: /* double_mat_rows is a tuple, its first
    element is an array of arrays, and its second element is an
    tuple representing its dimensions */
    double_mat_row { [| fst $1 |], (1, snd $1) }
  | double_mat_rows SEMI double_mat_row { Array.append (fst $1)
      [| fst $3 |], (fst (snd $1) + 1,snd (snd $1)) }

double_mat_row:
    element { [| $1 |], 1 }
  | double_mat_row COMMA element { Array.append (fst $1) [| $3
      |], snd $1 + 1 }

element:
    DOUBLE_LITERAL { $1 }
  | MINUS DOUBLE_LITERAL %prec NEG { -. $2 }
```

8.4 AST

```
(* Abstract Syntax Tree and functions for printing it *)

type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq |
    Greater | Geq | And | Or | Rmdr | Matprod | Filter

type uop = Neg | Not

type typ = Int | Bool | Image | Double | Matrix | Void | String
     | Mulret of typ list
```

```
type bind = typ * string

type expr =
    IntLit of int
  | StringLit of string
  | DoubleLit of float
  | BoolLit of bool
  | MatrixLit of float array array * (int * int)
  | Id of string
  | Binop of expr * op * expr
  | Comma of expr list
  | Unop of uop * expr
  | Assign of expr * expr
  | Mulassign of expr * expr
  | Index of string * (expr * expr)
  | Call of string * expr list
  | Noexpr
  | Noassign
  | Bug (* debug entity, not for other use *)
  | Range of index * index
and index = Beg | End | ExprInd of expr


type stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt
  | Local of typ * string * expr

type func_decl = {
    mutable typ : typ;
    fname : string;
    formals : bind list;
    body : stmt list;
  }


type program = func_decl list * stmt list
```

```
(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"
  | _ -> ""

let string_of_uop = function
    Neg -> "-"
  | Not -> "!"

let rec string_of_expr = function
    IntLit(l) -> string_of_int l
  | DoubleLit(l) -> string_of_float l
  | StringLit(l) -> l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | Id(s) -> s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^
          string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(v, e) -> string_of_expr v ^ " = " ^ string_of_expr e
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^
          ")"
  | Noexpr -> ""
  | _ -> ""

let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^
          "}\n"
```

```
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^
     string_of_stmt s
  | If(e, s1, s2) -> "if (" ^ string_of_expr e ^ ")\n" ^
     string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | For(e1, e2, e3, s) ->
     "for (" ^ string_of_expr e1 ^ " ; " ^ string_of_expr e2 ^ "
        ; " ^
     string_of_expr e3 ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^
     string_of_stmt s
  | _ -> ""

let string_of_typ = function
    Int -> "int"
  | Bool -> "bool"
  | Void -> "void"
  | Double -> "double"
  | Image -> "image"
  | Matrix -> "matrix"
  | String -> "string"
  | _ -> ""

let string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";\n"

let string_of_fdecl fdecl =
  string_of_typ fdecl.typ ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map snd
     fdecl.formals) ^
  ")\n{\n" ^
  (*String.concat "" (List.map string_of_vdecl fdecl.locals) ^*)
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "}\n"

let string_of_program (vars, funcs) =
  String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl funcs)
```

8.5 Semant

```
(* Semantic checking for the MicroC compiler *)

open Ast

module StringMap = Map.Make(String)

(* Semantic checking of a program. Returns void if successful,
   throws an exception if something is wrong. *)

let check (functions, _) =

  (* Raise an exception if the given list has a duplicate *)
  let report_duplicate exceptf list =
    let rec helper = function
   n1 :: n2 :: _ when n1 = n2 -> raise (Failure (exceptf n1))
      | _ :: t -> helper t
      | [] -> ()
    in helper (List.sort compare list)
  in

  (* Raise an exception if a given binding is to a void type *)
  let check_not_void exceptf = function
      (Void, n) -> raise (Failure (exceptf n))
    | _ -> ()
  in


  (**** Checking Functions ****)

  (* check built-in functions names are not used by users *)
  let report_built_in_duplicate list =
    let rec helper = function
      "size" :: _ -> raise (Failure ("Semantic error : name size
         is reserved."))
    | "zeros" :: _ -> raise (Failure ("Semantic error : name
       zeros is reserved."))
    | "double2int" :: _ -> raise (Failure ("Semantic error : name
       double2int is reserved."))
    | "int2double" :: _ -> raise (Failure ("Semantic error : name
       int2double is reserved."))
    | "load_cpp" :: _ -> raise (Failure ("Semantic error : name
       load_cpp is reserved."))
```

```
    | "load" :: _ -> raise (Failure ("Semantic error : name load
      is reserved."))
    | "save_cpp" :: _ -> raise (Failure ("Semantic error : name
      save_cpp is reserved."))
    | "save" :: _ -> raise (Failure ("Semantic error : name save
      is reserved."))
    | "faceDetect" :: _ -> raise (Failure ("Semantic error : name
      faceDetect is reserved."))
    | "face" :: _ -> raise (Failure ("Semantic error : name save
      is reserved."))
    | _ :: t -> helper t
    | [] -> ()
    in helper list
  in
  report_built_in_duplicate (List.map (fun fd -> fd.fname)
     functions);

  report_duplicate (fun n -> "Semantic error : duplicate
     function " ^ n)
    (List.map (fun fd -> fd.fname) functions);

  let check_function func =

    List.iter (check_not_void (fun n -> "illegal void formal " ^
      n ^
     " in " ^ func.fname)) func.formals;

    report_duplicate (fun n -> "duplicate formal " ^ n ^ " in " ^
      func.fname)
     (List.map snd func.formals);

  in
  List.iter check_function functions
```

8.6 codegen

```
(*!!! the format is a bit messed up in latex, you are advised to
   read codegen from our source file *)
(* Code generation: translate takes a semantically checked AST
   and
produces LLVM IR
```

```

LLVM tutorial: Make sure to read the OCaml version of the
   tutorial

http://llvm.org/docs/tutorial/index.html

Detailed documentation on the OCaml LLVM library:

http://llvm.moe/
http://llvm.moe/ocaml/

*)

module L = Llvm
module A = Ast
module H = Hashtbl
module StringMap = Map.Make(String)
type ret_typ = Returnstruct of L.lltype | Lltypearray of
   L.lltype array | Voidtype of L.lltype | Maintype
type access_link = Access of access_link * (string, L.llvalue)
   H.t | Null
let translate (functions, main_stmt) =

(*/* sample code structure
 1. default value: int:0 ; double:0. ; bool:true ; string:"" ;
    matrix:[]
 2. matrix operation:
    for each operation below: matrix dimension must agree

    i). matrix number element-wise : matrix op number | number
       op matrix (op : + - * / )
    ii). matrix matrix element-wise : matrix op matrix (op : +
       - * / $)
    iii). matrix product : matrix .* matrix
    iv). matrix indexing : matrix[x1, y1] | matrix[x1:x2,
       y1:y2] | matrix[x1:, y1] | matrix[:, y1] | matrix[:,
       :y2] | etc. basically the syntax of Matlab.
    v). matrix assignment : m1 = m2[x1:x2, y1:y2] | m1[x:, :y]
       = m2[x1:x2, y1:y2] | etc.
    vi). matrix equality and inequality : m1 == m2 | m1[x1:, :]
       != m2[x2:x3, y1:y2] | etc.
 3. built-in functions :
    i). size : syntax : i, j = size(m), return size of a matrix.
```

```
    ii). zeros: syntax : zeros(i, j), return a zero matrix of size i by j.
    iii). int2double : syntax : int2double(i), convert an int to double.
    iv). double2int : syntax : double2int(d), convert a double to int.
    v). save(m_r, m_g, m_b, path) : save image to path.
    vi). m_r, m_g, m_b = load(path) : load image.
    vii). m = face(path) : detect faces in the image at given path, return m is a 4 by n matrix, n is the number of faces, row 1 stores coordinates of the center of faces at which row, row 2 stores coordinates of the center of faces at which col, row 3 stores the height of the faces, row 4 stores the length of faces.
  4. std functions:
  5. error messages:
    i). Compiler error : used for debug purpose, it is very unlikely that user would see any of them.
    ii). Syntax error : followed by a description on the error.
    iii). Semantic error : followed by description on the error.
func f1(...) { return;}
func f2(matrix m, int i, double d, string s) { return m1, m2, d1, s1;}
matrix m1 = [1.0,2.0;3.0,4.0];
matrix m2;
double d1 =3.4;
string s;
m1[1:,:], m2, d1, s = f2([1.0;3.0], 5, 2.3, "facelab");
*/*)


(* 1. Auxiliary definitions *)
 let context = L.global_context () in
 let the_module = L.create_module context "Facelab"
 and double_t = L.double_type context
 and i32_t = L.i32_type context
 and i8_t = L.i8_type context in
 let str_t = L.pointer_type i8_t
 and i1_t = L.i1_type context
 and void_t = L.void_type context in
 let matrix_t = L.named_struct_type context "matrix_t" in
 L.struct_set_body matrix_t [|L.pointer_type double_t; i32_t; i32_t|] false;
```

```

(* declare main first, so that some of the global variables can be stored in the stack of main. Its body will be populated in later section *)

  let main_name = "main" in
  let main_define = (* main_define the "the_function" equivalent of main function *)
    let main_formal = [| |] in (* empty array *)
    let main_type = L.function_type i32_t main_formal in
    L.define_function main_name main_type the_module in
  let main_builder = ref (* main_builder the "builder" equivalent of main function *)
    (L.builder_at_end context (L.entry_block main_define)) in

  let function_decls = H.create (List.length functions + 1000) in


  (* AST.expr type to LLVM type conversion *)
  let ltype_of_typ = function
      A.Int -> i32_t
    | A.Double -> double_t
    | A.String -> str_t (* pointer to store string *)
    | A.Bool -> i1_t
    | A.Void -> void_t
    | A.Matrix -> matrix_t
    | _ -> failwith("Compiler error : ltype_of_typ function matching error.")
  in

  let type_of_lltype typ =
    let ltype_string = L.string_of_lltype typ in
    match ltype_string with
      "void" -> A.Void
    | "i32" -> A.Int
    | "double" -> A.Double
    | "i1" -> A.Bool
    | "i8*" -> A.String
    | "%matrix_t*" -> A.Matrix
    | _ -> failwith("Compiler error : type_of_lltype function matching error.")
  in

```

```
let typ_of_lvalue lv =
 let lltype = L.type_of lv in
 type_of_lltype lltype
in

let is_matrix ptr =
 let ltype_string = L.string_of_lltype (L.type_of ptr) in
 match ltype_string with
   "%matrix_t*" -> true
  | _ -> false
in

(* Declare printf(), which the print built-in function will call *)
let printf_t = L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
let printf_func = L.declare_function "printf" printf_t the_module in

(* use to interrupt the function flow and throw run-time exception *)
let abort_func = L.declare_function "abort" (L.function_type void_t [||]) the_module in

(* Invoke "f builder" if the current block does not already
    have a terminal (e.g., a branch). *)
let add_terminal builder f =
 match L.block_terminator (L.insertion_block !builder) with (* block terminator is one of the following in a block : ret, br, switch, indirectbr, invoke, unwind, unreachable*)
   Some _ -> () (* Some a ocaml construct matching with a not null set, None match a null set *)
  | None -> ignore (f !builder)
in

(* format strings *)
let string_format_str = L.build_global_stringptr "%s" "fmt_str" !main_builder in
let double_format_str = L.build_global_stringptr "%f" "fmt_double" !main_builder in
let int_format_str = L.build_global_stringptr "%d" "fmt_int" !main_builder in
```

```
let new_line_str = L.build_global_stringptr "\n" "fmt_str" !main_builder in
let two_space_str = L.build_global_stringptr "  " "fmt_str" !main_builder in
let empty_str = L.build_global_stringptr "" "fmt_str" !main_builder in
let true_str = L.build_global_stringptr "true" "fmt_str" !main_builder in
let false_str = L.build_global_stringptr "false" "fmt_str" !main_builder in
let mat_dim_err_str = L.build_global_stringptr "Semantic error : wrong dimension of operands of matrix operation." "fmt_str" !main_builder in
let mat_bound_err_str = L.build_global_stringptr "Semantic error : matrix index out of bounds." "fmt_str" !main_builder in
let mat_assign_err_str = L.build_global_stringptr "Semantic error : matrix block assignment must have agreeable dimension on both sides." "fmt_str" !main_builder in


(* following function builds llvm control flow *)
(* llvm if *)
let llvm_if function_ptr builder (predicate, then_stmt, else_stmt) =
  let merge_bb = L.append_block context "merge" function_ptr in (* "merge" is something like an entry, so are the rest *)

  let then_bb = L.append_block context "then" function_ptr in
  let then_builder = ref (L.builder_at_end context then_bb) in
  add_terminal (then_stmt then_builder) (L.build_br merge_bb); (* L.build_br syntax : br entry *)

  let else_bb = L.append_block context "else" function_ptr in
  let else_builder = ref (L.builder_at_end context else_bb) in
  add_terminal (else_stmt else_builder) (L.build_br merge_bb);

  let bool_val = predicate builder in
  ignore (L.build_cond_br bool_val then_bb else_bb !builder); (* L.build_cond_br syntax : br bool entry1 entry2 *)
  let merge_builder = ref (L.builder_at_end context merge_bb) in
  builder := !merge_builder; merge_builder
in
```

```
  (* llvm while *)
  let llvm_while function_ptr builder (predicate, body_stmt) =
    let pred_bb = L.append_block context "while" function_ptr in
    let pred_builder = ref (L.builder_at_end context pred_bb) in
    ignore (L.build_br pred_bb !builder);

    let body_bb = L.append_block context "while_body" function_ptr in
    let body_builder = ref (L.builder_at_end context body_bb) in
    add_terminal (body_stmt body_builder)
    (L.build_br pred_bb);

    let merge_bb = L.append_block context "merge" function_ptr in

    let bool_var = predicate pred_builder in
    ignore (L.build_cond_br bool_var body_bb merge_bb !pred_builder);
    let merge_builder = ref (L.builder_at_end context merge_bb) in
    builder := !merge_builder; merge_builder
  in
  (* llvm for *)
  let llvm_for function_ptr builder (init, predicate, update, body_stmt) =
    ignore(init builder);
    let combined_stmt builder = body_stmt builder; update builder in
    llvm_while function_ptr builder (predicate, combined_stmt)
  in
(* matrix auxiliaries *)

  (* access an entries in a matrix *)
  let access mat r c x y builder =
    ignore(r); (* no use but suppress warning *)
    let index = L.build_add y (L.build_mul c x "tmp" !builder) "index" !builder in
    L.build_gep mat [|index|] "element_ptr" !builder
  in

  (* matrix literal building helper *)
  let build_mat_lit (v, (r,c)) builder=
    let mat = L.build_array_alloca double_t (L.const_int i32_t (r*c)) "system_mat" !builder in
    (for i = 0 to (r-1) do
```

```
      for j = 0 to (c-1) do
        let element_ptr = access mat (L.const_int i32_t r) (L.const_int i32_t c) (L.const_int i32_t i) (L.const_int i32_t j) builder in
        ignore(L.build_store (L.const_float double_t v.(i).(j)) element_ptr !builder)
      done
    done);
    let m = L.build_alloca matrix_t "m" !builder in
    let m_mat = L.build_struct_gep m 0 "m_mat" !builder in
    ignore(L.build_store mat m_mat !builder);
    let m_r = L.build_struct_gep m 1 "m_r" !builder in
    ignore(L.build_store (L.const_int i32_t r) m_r !builder);
    let m_c = L.build_struct_gep m 2 "m_c" !builder in
    ignore(L.build_store (L.const_int i32_t c) m_c !builder); m
  in

  (* create a matrix of size r by c (where r c are llvalues) *)
  let build_mat_init alloc_func array_alloc_func r c function_ptr builder =
    let size = L.build_mul r c "size" !builder in
    let mat = array_alloc_func double_t size "system_mat" !builder in
    let m = alloc_func matrix_t "m" !builder in
    let m_mat = L.build_struct_gep m 0 "m_mat" !builder in
    ignore(L.build_store mat m_mat !builder);
    let m_r = L.build_struct_gep m 1 "m_r" !builder in
    ignore(L.build_store r m_r !builder);
    let m_c = L.build_struct_gep m 2 "m_c" !builder in
    ignore(L.build_store c m_c !builder);
    let r_high = L.build_sub r (L.const_int i32_t 1) "tmp" !builder in
    let c_high = L.build_sub c (L.const_int i32_t 1) "tmp" !builder in
    (*IMPORTANT: initialize to 0, otherwise it will start with some garbage value, and therefore give wrong results.*)
    let i = L.build_alloca i32_t "i" !builder in
    let init_i builder = L.build_store (L.const_int i32_t 0) i !builder in
    let predicate_i builder = L.build_icmp L.Icmp.Sle (L.build_load i "i_v" !builder) r_high "bool_val" !builder in
```

```
    let update_i builder = ignore(L.build_store (L.build_add (L.build_load i "i_v" !builder) (L.const_int i32_t 1) "tmp" !builder) i !builder);builder in
    let body_stmt_i builder =
      let j = L.build_alloca i32_t "j" !builder in
      let init_j builder = L.build_store (L.const_int i32_t 0) j !builder in
      let predicate_j builder = L.build_icmp L.Icmp.Sle (L.build_load j "j_v" !builder) c_high "bool_val" !builder in
      let update_j builder = ignore(L.build_store (L.build_add (L.build_load j "j_v" !builder) (L.const_int i32_t 1) "tmp" !builder) j !builder);builder in
      let body_stmt_j builder =
        let mat_element_ptr = access mat r c (L.build_load i "i_v" !builder) (L.build_load j "j_v" !builder) builder in
        ignore(L.build_store (L.const_float double_t 0.0) mat_element_ptr !builder) in
      ignore(llvm_for function_ptr builder (init_j, predicate_j, update_j, body_stmt_j)) in
    ignore(llvm_for function_ptr builder (init_i, predicate_i, update_i, body_stmt_i));m
  in
  let stack_build_mat_init r c function_ptr builder =
    build_mat_init L.build_alloca L.build_array_alloca r c function_ptr builder in
  let heap_build_mat_init r c function_ptr builder =
    build_mat_init L.build_malloc L.build_array_malloc r c function_ptr builder in

  (* assign an array to an array on the stack *)
  let mat_assign m_mat x_low x_high y_low y_high v_mat v_x_low v_y_low function_ptr builder =
    let mat = L.build_load (L.build_struct_gep m_mat 0 "m_mat" !builder) "mat_mat" !builder in
    let r_mat = L.build_load (L.build_struct_gep m_mat 1 "m_r" !builder) "r_mat" !builder in
    let c_mat = L.build_load (L.build_struct_gep m_mat 2 "m_c" !builder) "c_mat" !builder in
    let v = L.build_load (L.build_struct_gep v_mat 0 "m_mat" !builder) "mat_v" !builder in
    let r_v = L.build_load (L.build_struct_gep v_mat 1 "m_r" !builder) "r_v" !builder in
```

```
    let c_v = L.build_load (L.build_struct_gep v_mat 2 "m_c"
        !builder) "c_v" !builder in
    let i = L.build_alloca i32_t "i" !builder in
    let init_i builder = L.build_store x_low i !builder in
    let predicate_i builder = L.build_icmp L.Icmp.Sle
        (L.build_load i "i_v" !builder) x_high "bool_val" !builder
        in
    let update_i builder = ignore(L.build_store (L.build_add
        (L.build_load i "i_v" !builder) (L.const_int i32_t 1)
        "tmp" !builder) i !builder);builder in
    let body_stmt_i builder =
      let j = L.build_alloca i32_t "j" !builder in
      let init_j builder = L.build_store y_low j !builder in
      let predicate_j builder = L.build_icmp L.Icmp.Sle
          (L.build_load j "j_v" !builder) y_high "bool_val"
          !builder in
      let update_j builder = ignore(L.build_store (L.build_add
          (L.build_load j "j_v" !builder) (L.const_int i32_t 1)
          "tmp" !builder) j !builder);builder in
      let body_stmt_j builder =
        let mat_element_ptr = access mat r_mat c_mat (L.build_load
            i "i_v" !builder) (L.build_load j "j_v" !builder)
            builder in
        let v_element_ptr = access v r_v c_v (L.build_add
            (L.build_sub (L.build_load i "i_v" !builder) x_low
            "tmp" !builder) v_x_low "tmp" !builder)
                                    (L.build_add (L.build_sub
                                        (L.build_load j "j_v"
                                        !builder) y_low "tmp"
                                        !builder) v_y_low "tmp"
                                        !builder) builder in
        let tmp_element = L.build_load v_element_ptr "tmp_element"
            !builder in
        ignore(L.build_store tmp_element mat_element_ptr !builder)
            in
      llvm_for function_ptr builder (init_j, predicate_j,
          update_j, body_stmt_j) in
    llvm_for function_ptr builder (init_i, predicate_i, update_i,
        body_stmt_i)
  in

  (* print an array *)
  let mat_print m_mat function_ptr builder=
```

```
let mat = L.build_load (L.build_struct_gep m_mat 0 "m_mat"
   !builder) "mat_mat" !builder in
let r_mat = L.build_load (L.build_struct_gep m_mat 1 "m_r"
   !builder) "r_mat" !builder in
let c_mat = L.build_load (L.build_struct_gep m_mat 2 "m_c"
   !builder) "c_mat" !builder in
let r'_mat = L.build_sub r_mat (L.const_int i32_t 1) "tmp"
   !builder in
let c'_mat = L.build_sub c_mat (L.const_int i32_t 1) "tmp"
   !builder in
let i = L.build_alloca i32_t "i" !builder in
let init_i builder = L.build_store (L.const_int i32_t 0) i
   !builder in
let predicate_i builder = L.build_icmp L.Icmp.Sle
   (L.build_load i "i_v" !builder) r'_mat "bool_val" !builder
   in
let update_i builder = ignore(L.build_store (L.build_add
   (L.build_load i "i_v" !builder) (L.const_int i32_t 1)
   "tmp" !builder) i !builder);builder in
let body_stmt_i builder =
  let j = L.build_alloca i32_t "j" !builder in
  let init_j builder = L.build_store (L.const_int i32_t 0) j
     !builder in
  let predicate_j builder = L.build_icmp L.Icmp.Sle
     (L.build_load j "j_v" !builder) c'_mat "bool_val"
     !builder in
  let update_j builder = ignore(L.build_store (L.build_add
     (L.build_load j "j_v" !builder) (L.const_int i32_t 1)
     "tmp" !builder) j !builder);builder in
  let body_stmt_j builder =
    let mat_element_ptr = access mat r_mat c_mat (L.build_load
       i "i_v" !builder) (L.build_load j "j_v" !builder)
       builder in
    let tmp_element = L.build_load mat_element_ptr
       "tmp_element" !builder in
    ignore(L.build_call printf_func [| double_format_str ;
       tmp_element|] "printf" !builder);
    ignore(L.build_call printf_func [| string_format_str ;
       two_space_str |] "printf" !builder) in
  ignore(llvm_for function_ptr builder (init_j, predicate_j,
     update_j, body_stmt_j));
  ignore(L.build_call printf_func [| string_format_str ;
     new_line_str |] "printf" !builder) in
```

```
    ignore(llvm_for function_ptr builder (init_i, predicate_i, update_i, body_stmt_i));
    L.build_call printf_func [| string_format_str ; empty_str |] "printf" !builder
  in


(* matrix matrix element wise operation *)
  let mat_mat_element_wise m1_mat m2_mat operator function_ptr builder=
    let m1 = L.build_load (L.build_struct_gep m1_mat 0 "m_mat" !builder) "mat_mat" !builder in
    let r = L.build_load (L.build_struct_gep m1_mat 1 "m_r" !builder) "r_mat" !builder in
    let r_high = L.build_sub r (L.const_int i32_t 1) "tmp" !builder in
    let c = L.build_load (L.build_struct_gep m1_mat 2 "m_c" !builder) "c_mat" !builder in
    let c_high = L.build_sub c (L.const_int i32_t 1) "tmp" !builder in
    let m2 = L.build_load (L.build_struct_gep m2_mat 0 "m_mat" !builder) "mat_v" !builder in
    let result_mat = stack_build_mat_init r c function_ptr builder in
    let result = L.build_load (L.build_struct_gep result_mat 0 "m_mat" !builder) "mat_mat" !builder in
    let i = L.build_alloca i32_t "i" !builder in
    let init_i builder = L.build_store (L.const_int i32_t 0) i !builder in
    let predicate_i builder = L.build_icmp L.Icmp.Sle (L.build_load i "i_v" !builder) r_high "bool_val" !builder in
    let update_i builder = ignore(L.build_store (L.build_add (L.build_load i "i_v" !builder) (L.const_int i32_t 1) "tmp" !builder) i !builder);builder in
    let body_stmt_i builder =
      let j = L.build_alloca i32_t "j" !builder in
      let init_j builder = L.build_store (L.const_int i32_t 0) j !builder in
      let predicate_j builder = L.build_icmp L.Icmp.Sle (L.build_load j "j_v" !builder) c_high "bool_val" !builder in
```

```
      let update_j builder = ignore(L.build_store (L.build_add (L.build_load j "j_v" !builder) (L.const_int i32_t 1) "tmp" !builder) j !builder);builder in
      let body_stmt_j builder =
        let m1_element_ptr = access m1 r c (L.build_load i "i_v" !builder) (L.build_load j "j_v" !builder) builder in
        let m1_element = L.build_load m1_element_ptr "tmp_element" !builder in
        let m2_element_ptr = access m2 r c (L.build_load i "i_v" !builder) (L.build_load j "j_v" !builder) builder in
        let m2_element = L.build_load m2_element_ptr "tmp_element" !builder in
        let result_element_ptr = access result r c (L.build_load i "i_v" !builder) (L.build_load j "j_v" !builder) builder in
        let tmp_element = operator m1_element m2_element "tmp_element" !builder in
        ignore(L.build_store tmp_element result_element_ptr !builder) in
      ignore(llvm_for function_ptr builder (init_j, predicate_j, update_j, body_stmt_j)) in
    ignore(llvm_for function_ptr builder (init_i, predicate_i, update_i, body_stmt_i)); result_mat
  in

  (*matrix equality *)
  let mat_equal m1_mat m2_mat function_ptr builder=
    let m1 = L.build_load (L.build_struct_gep m1_mat 0 "m_mat" !builder) "mat_mat" !builder in
    let r = L.build_load (L.build_struct_gep m1_mat 1 "m_r" !builder) "r_mat" !builder in
    let r_high = L.build_sub r (L.const_int i32_t 1) "tmp" !builder in
    let c = L.build_load (L.build_struct_gep m1_mat 2 "m_c" !builder) "c_mat" !builder in
    let c_high = L.build_sub c (L.const_int i32_t 1) "tmp" !builder in
    let m2 = L.build_load (L.build_struct_gep m2_mat 0 "m_mat" !builder) "mat_v" !builder in
    let result = L.build_alloca i1_t "result" !builder in
    ignore(L.build_store (L.const_int i1_t 1) result !builder);
    let i = L.build_alloca i32_t "i" !builder in
```

```
    let init_i builder = L.build_store (L.const_int i32_t 0) i
       !builder in
    let predicate_i builder = L.build_icmp L.Icmp.Sle
       (L.build_load i "i_v" !builder) r_high "bool_val" !builder
       in
    let update_i builder = ignore(L.build_store (L.build_add
       (L.build_load i "i_v" !builder) (L.const_int i32_t 1)
       "tmp" !builder) i !builder);builder in
    let body_stmt_i builder =
      let j = L.build_alloca i32_t "j" !builder in
      let init_j builder = L.build_store (L.const_int i32_t 0) j
         !builder in
      let predicate_j builder = L.build_icmp L.Icmp.Sle
         (L.build_load j "j_v" !builder) c_high "bool_val"
         !builder in
      let update_j builder = ignore(L.build_store (L.build_add
         (L.build_load j "j_v" !builder) (L.const_int i32_t 1)
         "tmp" !builder) j !builder);builder in
      let body_stmt_j builder =
        let m1_element_ptr = access m1 r c (L.build_load i "i_v"
           !builder) (L.build_load j "j_v" !builder) builder in
        let m1_element = L.build_load m1_element_ptr "tmp_element"
           !builder in
        let m2_element_ptr = access m2 r c (L.build_load i "i_v"
           !builder) (L.build_load j "j_v" !builder) builder in
        let m2_element = L.build_load m2_element_ptr "tmp_element"
           !builder in
        let predicate builder = L.build_fcmp L.Fcmp.One m1_element
           m2_element "tmp" !builder in
        let then_stmt builder = ignore(L.build_store (L.const_int
           i1_t 0) result !builder); builder in
        let else_stmt builder = builder in
        ignore(llvm_if function_ptr builder (predicate, then_stmt,
           else_stmt)) in
      ignore(llvm_for function_ptr builder (init_j, predicate_j,
         update_j, body_stmt_j)) in
    ignore(llvm_for function_ptr builder (init_i, predicate_i,
       update_i, body_stmt_i));
    L.build_load result "result" !builder
  in

  let mat_not_equal m1_mat m2_mat function_ptr builder=
    let result = L.build_alloca i1_t "result" !builder in
```

```
    let tmp = mat_equal m1_mat m2_mat function_ptr builder in
    let predicate builder = L.build_icmp L.Icmp.Ne tmp
       (L.const_int i1_t 1) "tmp" !builder in
    let then_stmt builder = ignore(L.build_store (L.const_int
       i1_t 1) result !builder); builder in
    let else_stmt builder = ignore(L.build_store (L.const_int
       i1_t 0) result !builder); builder in
    ignore(llvm_if function_ptr builder (predicate, then_stmt,
       else_stmt));
    L.build_load result "result" !builder
  in

  (* matrix number element wise operation *)
  let mat_num_element_wise m1_mat num operator function_ptr
     builder=
    let m1 = L.build_load (L.build_struct_gep m1_mat 0 "m_mat"
       !builder) "mat_mat" !builder in
    let r = L.build_load (L.build_struct_gep m1_mat 1 "m_r"
       !builder) "r_mat" !builder in
    let r_high = L.build_sub r (L.const_int i32_t 1) "tmp"
       !builder in
    let c = L.build_load (L.build_struct_gep m1_mat 2 "m_c"
       !builder) "c_mat" !builder in
    let c_high = L.build_sub c (L.const_int i32_t 1) "tmp"
       !builder in
    let result_mat = stack_build_mat_init r c function_ptr
       builder in
    let result = L.build_load (L.build_struct_gep result_mat 0
       "m_mat" !builder) "mat_mat" !builder in
    let i = L.build_alloca i32_t "i" !builder in
    let init_i builder = L.build_store (L.const_int i32_t 0) i
       !builder in
    let predicate_i builder = L.build_icmp L.Icmp.Sle
       (L.build_load i "i_v" !builder) r_high "bool_val" !builder
       in
    let update_i builder = ignore(L.build_store (L.build_add
       (L.build_load i "i_v" !builder) (L.const_int i32_t 1)
       "tmp" !builder) i !builder);builder in
    let body_stmt_i builder =
      let j = L.build_alloca i32_t "j" !builder in
      let init_j builder = L.build_store (L.const_int i32_t 0) j
         !builder in
```

```
    let predicate_j builder = L.build_icmp L.Icmp.Sle (L.build_load j "j_v" !builder) c_high "bool_val" !builder in
    let update_j builder = ignore(L.build_store (L.build_add (L.build_load j "j_v" !builder) (L.const_int i32_t 1) "tmp" !builder) j !builder);builder in
    let body_stmt_j builder =
     let m1_element_ptr = access m1 r c (L.build_load i "i_v" !builder) (L.build_load j "j_v" !builder) builder in
     let m1_element = L.build_load m1_element_ptr "tmp_element" !builder in
     let result_element_ptr = access result r c (L.build_load i "i_v" !builder) (L.build_load j "j_v" !builder) builder in
     let tmp_element = operator m1_element num "tmp_element" !builder in
     ignore(L.build_store tmp_element result_element_ptr !builder) in
    ignore(llvm_for function_ptr builder (init_j, predicate_j, update_j, body_stmt_j)) in
  ignore(llvm_for function_ptr builder (init_i, predicate_i, update_i, body_stmt_i)); result_mat
 in


 (*matrix product*)
 let mat_mat_product m1_mat m2_mat function_ptr builder=
  let m1 = L.build_load (L.build_struct_gep m1_mat 0 "m_mat" !builder) "mat_mat" !builder in
  let m2 = L.build_load (L.build_struct_gep m2_mat 0 "m_mat" !builder) "mat_v" !builder in
  let r = L.build_load (L.build_struct_gep m1_mat 1 "m_r" !builder) "r_mat" !builder in
  let r_high = L.build_sub r (L.const_int i32_t 1) "tmp" !builder in
  let c = L.build_load (L.build_struct_gep m2_mat 2 "m_c" !builder) "c_mat" !builder in
  let c_high = L.build_sub c (L.const_int i32_t 1) "tmp" !builder in
  let l = L.build_load (L.build_struct_gep m1_mat 2 "m_l" !builder) "l_mat" !builder in
  let l_high = L.build_sub l (L.const_int i32_t 1) "tmp" !builder in
```

```
    let result_mat = stack_build_mat_init r c function_ptr
       builder in
    let result = L.build_load (L.build_struct_gep result_mat 0
       "m_mat" !builder) "mat_mat" !builder in
    let i = L.build_alloca i32_t "i" !builder in
    let init_i builder = L.build_store (L.const_int i32_t 0) i
       !builder in
    let predicate_i builder = L.build_icmp L.Icmp.Sle
       (L.build_load i "i_v" !builder) r_high "bool_val" !builder
       in
    let update_i builder = ignore(L.build_store (L.build_add
       (L.build_load i "i_v" !builder) (L.const_int i32_t 1)
       "tmp" !builder) i !builder);builder in
    let body_stmt_i builder =
      let j = L.build_alloca i32_t "j" !builder in
      let init_j builder = L.build_store (L.const_int i32_t 0) j
         !builder in
      let predicate_j builder = L.build_icmp L.Icmp.Sle
         (L.build_load j "j_v" !builder) c_high "bool_val"
         !builder in
      let update_j builder = ignore(L.build_store (L.build_add
         (L.build_load j "j_v" !builder) (L.const_int i32_t 1)
         "tmp" !builder) j !builder);builder in
      let body_stmt_j builder =
        let result_element_ptr = access result r c (L.build_load i
           "i_v" !builder) (L.build_load j "j_v" !builder) builder
           in
        let tmp_element = L.build_alloca double_t "tmp_element"
           !builder in
        ignore(L.build_store (L.const_float double_t 0.0)
           tmp_element !builder); (*IMPORTANT: initialize to 0,
           otherwise it will start with some garbage value, and
           therefore give wrong results.*)
        let k = L.build_alloca i32_t "k" !builder in
        let init_k builder = L.build_store (L.const_int i32_t 0) k
           !builder in
        let predicate_k builder = L.build_icmp L.Icmp.Sle
           (L.build_load k "k_v" !builder) l_high "bool_val"
           !builder in
        let update_k builder = ignore(L.build_store (L.build_add
           (L.build_load k "k_v" !builder) (L.const_int i32_t 1)
           "tmp" !builder) k !builder);builder in
        let body_stmt_k builder =
```

```
        let m1_element_ptr = access m1 r l (L.build_load i "i_v" !builder) (L.build_load k "k_v" !builder) builder in
        let m1_element = L.build_load m1_element_ptr "tmp_element" !builder in
        let m2_element_ptr = access m2 l c (L.build_load k "k_v" !builder) (L.build_load j "j_v" !builder) builder in
        let m2_element = L.build_load m2_element_ptr "tmp_element" !builder in
        ignore(L.build_store (L.build_fadd (L.build_fmul m1_element m2_element "tmp" !builder) (L.build_load tmp_element "tmp" !builder) "tmp" !builder) tmp_element !builder) in
      ignore(llvm_for function_ptr builder (init_k, predicate_k, update_k, body_stmt_k));
      ignore(L.build_store (L.build_load tmp_element "tmp" !builder) result_element_ptr !builder) in
    ignore(llvm_for function_ptr builder (init_j, predicate_j, update_j, body_stmt_j)) in
  ignore(llvm_for function_ptr builder (init_i, predicate_i, update_i, body_stmt_i)); result_mat
in

(* rgb array to rgb matrix *)
let to_rgb_matrix mat_arr mat_r mat_g mat_b r c function_ptr builder =
  let m_r = L.build_load (L.build_struct_gep mat_r 0 "mat_r" !builder) "mat_mat" !builder in
  let m_g = L.build_load (L.build_struct_gep mat_g 0 "mat_g" !builder) "mat_mat" !builder in
  let m_b = L.build_load (L.build_struct_gep mat_b 0 "mat_b" !builder) "mat_mat" !builder in
  let r_high = L.build_sub r (L.const_int i32_t 1) "tmp" !builder in
  let c_high = L.build_sub c (L.const_int i32_t 1) "tmp" !builder in
  let counter = L.build_alloca i32_t "counter" !builder in
  ignore(L.build_store (L.const_int i32_t 2) counter !builder);
  let i = L.build_alloca i32_t "i" !builder in
  let init_i builder = L.build_store (L.const_int i32_t 0) i !builder in
  let predicate_i builder = L.build_icmp L.Icmp.Sle (L.build_load i "i_v" !builder) r_high "bool_val" !builder in
```

```
    let update_i builder = ignore(L.build_store (L.build_add (L.build_load i "i_v" !builder) (L.const_int i32_t 1) "tmp" !builder) i !builder);builder in
    let body_stmt_i builder =
      let j = L.build_alloca i32_t "j" !builder in
      let init_j builder = L.build_store (L.const_int i32_t 0) j !builder in
      let predicate_j builder = L.build_icmp L.Icmp.Sle (L.build_load j "j_v" !builder) c_high "bool_val" !builder in
      let update_j builder = ignore(L.build_store (L.build_add (L.build_load j "j_v" !builder) (L.const_int i32_t 1) "tmp" !builder) j !builder);builder in
      let body_stmt_j builder =
        let m_r_element_ptr = access m_r r c (L.build_load i "i_v" !builder) (L.build_load j "j_v" !builder) builder in
        let m_g_element_ptr = access m_g r c (L.build_load i "i_v" !builder) (L.build_load j "j_v" !builder) builder in
        let m_b_element_ptr = access m_b r c (L.build_load i "i_v" !builder) (L.build_load j "j_v" !builder) builder in
        ignore(L.build_store (L.build_load (L.build_gep mat_arr [|(L.build_load counter "counter" !builder)|] "element_ptr" !builder) "tmp_element" !builder) m_b_element_ptr !builder);
        let tmp = L.build_add (L.build_load counter "counter" !builder) (L.const_int i32_t 1) "tmp" !builder in
        ignore(L.build_store tmp counter !builder);
        ignore(L.build_store (L.build_load (L.build_gep mat_arr [|(L.build_load counter "counter" !builder)|] "element_ptr" !builder) "tmp_element" !builder) m_g_element_ptr !builder);
        let tmp = L.build_add (L.build_load counter "counter" !builder) (L.const_int i32_t 1) "tmp" !builder in
        ignore(L.build_store tmp counter !builder);
        ignore(L.build_store (L.build_load (L.build_gep mat_arr [|(L.build_load counter "counter" !builder)|] "element_ptr" !builder) "tmp_element" !builder) m_r_element_ptr !builder);
        let tmp = L.build_add (L.build_load counter "counter" !builder) (L.const_int i32_t 1) "tmp" !builder in
        ignore(L.build_store tmp counter !builder) in
      ignore(llvm_for function_ptr builder (init_j, predicate_j, update_j, body_stmt_j)) in
```

```
    ignore(llvm_for function_ptr builder (init_i, predicate_i,
        update_i, body_stmt_i))
  in

(* rgb matrix to rgb array *)
  let from_rgb_matrix mat_arr mat_r mat_g mat_b r c function_ptr
      builder =
    let m_r = L.build_load (L.build_struct_gep mat_r 0 "mat_r"
        !builder) "mat_mat" !builder in
    let m_g = L.build_load (L.build_struct_gep mat_g 0 "mat_g"
        !builder) "mat_mat" !builder in
    let m_b = L.build_load (L.build_struct_gep mat_b 0 "mat_b"
        !builder) "mat_mat" !builder in
    let r_high = L.build_sub r (L.const_int i32_t 1) "tmp"
        !builder in
    let c_high = L.build_sub c (L.const_int i32_t 1) "tmp"
        !builder in
    ignore(L.build_store (L.build_sitofp r double_t "tmp"
        !builder) (L.build_gep mat_arr [|(L.const_int i32_t 0)|]
        "element_ptr" !builder) !builder);
    ignore(L.build_store (L.build_sitofp c double_t "tmp"
        !builder) (L.build_gep mat_arr [|(L.const_int i32_t 1)|]
        "element_ptr" !builder) !builder);
    let counter = L.build_alloca i32_t "counter" !builder in
    ignore(L.build_store (L.const_int i32_t 2) counter !builder);
    let i = L.build_alloca i32_t "i" !builder in
    let init_i builder = L.build_store (L.const_int i32_t 0) i
        !builder in
    let predicate_i builder = L.build_icmp L.Icmp.Sle
        (L.build_load i "i_v" !builder) r_high "bool_val" !builder
        in
    let update_i builder = ignore(L.build_store (L.build_add
        (L.build_load i "i_v" !builder) (L.const_int i32_t 1)
        "tmp" !builder) i !builder);builder in
    let body_stmt_i builder =
      let j = L.build_alloca i32_t "j" !builder in
      let init_j builder = L.build_store (L.const_int i32_t 0) j
          !builder in
      let predicate_j builder = L.build_icmp L.Icmp.Sle
          (L.build_load j "j_v" !builder) c_high "bool_val"
          !builder in
      let update_j builder = ignore(L.build_store (L.build_add
          (L.build_load j "j_v" !builder) (L.const_int i32_t 1)
```

```
        "tmp" !builder) j !builder);builder in
      let body_stmt_j builder =
        let m_r_element_ptr = access m_r r c (L.build_load i "i_v"
            !builder) (L.build_load j "j_v" !builder) builder in
        let m_g_element_ptr = access m_g r c (L.build_load i "i_v"
            !builder) (L.build_load j "j_v" !builder) builder in
        let m_b_element_ptr = access m_b r c (L.build_load i "i_v"
            !builder) (L.build_load j "j_v" !builder) builder in
        ignore(L.build_store (L.build_load m_b_element_ptr
            "tmp_element" !builder) (L.build_gep mat_arr
            [|(L.build_load counter "counter" !builder)|]
            "element_ptr" !builder) !builder);
        let tmp = L.build_add (L.build_load counter "counter"
            !builder) (L.const_int i32_t 1) "tmp" !builder in
        ignore(L.build_store tmp counter !builder);
        ignore(L.build_store (L.build_load m_g_element_ptr
            "tmp_element" !builder) (L.build_gep mat_arr
            [|(L.build_load counter "counter" !builder)|]
            "element_ptr" !builder) !builder);
        let tmp = L.build_add (L.build_load counter "counter"
            !builder) (L.const_int i32_t 1) "tmp" !builder in
        ignore(L.build_store tmp counter !builder);
        ignore(L.build_store (L.build_load m_r_element_ptr
            "tmp_element" !builder) (L.build_gep mat_arr
            [|(L.build_load counter "counter" !builder)|]
            "element_ptr" !builder) !builder);
        let tmp = L.build_add (L.build_load counter "counter"
            !builder) (L.const_int i32_t 1) "tmp" !builder in
        ignore(L.build_store tmp counter !builder) in
      ignore(llvm_for function_ptr builder (init_j, predicate_j,
        update_j, body_stmt_j)) in
    ignore(llvm_for function_ptr builder (init_i, predicate_i,
      update_i, body_stmt_i))
  in


  (* face array to face matrix *)
  let face_matrix mat_arr mat num function_ptr builder =
    let m = L.build_load (L.build_struct_gep mat 0 "mat_r"
      !builder) "mat_mat" !builder in
    let counter = L.build_alloca i32_t "counter" !builder in
    ignore(L.build_store (L.const_int i32_t 1) counter !builder);
```

```
    let num_high = L.build_sub num (L.const_int i32_t 1) "tmp"
       !builder in
    let i = L.build_alloca i32_t "i" !builder in
    let init_i builder = L.build_store (L.const_int i32_t 0) i
       !builder in
    let predicate_i builder = L.build_icmp L.Icmp.Sle
       (L.build_load i "i_v" !builder) num_high "bool_val"
       !builder in
    let update_i builder = ignore(L.build_store (L.build_add
       (L.build_load i "i_v" !builder) (L.const_int i32_t 1)
       "tmp" !builder) i !builder);builder in
    let body_stmt_i builder =
      let j = L.build_alloca i32_t "j" !builder in
      let init_j builder = L.build_store (L.const_int i32_t 0) j
         !builder in
      let predicate_j builder = L.build_icmp L.Icmp.Sle
         (L.build_load j "j_v" !builder) (L.const_int i32_t 3)
         "bool_val" !builder in
      let update_j builder = ignore(L.build_store (L.build_add
         (L.build_load j "j_v" !builder) (L.const_int i32_t 1)
         "tmp" !builder) j !builder);builder in
      let body_stmt_j builder =
        let m_element_ptr = access m (L.const_int i32_t 4) num
           (L.build_load i "i_v" !builder) (L.build_load j "j_v"
           !builder) builder in
        ignore(L.build_store (L.build_load (L.build_gep mat_arr
           [|(L.build_load counter "counter" !builder)|]
           "element_ptr" !builder) "tmp_element" !builder)
           m_element_ptr !builder);
        let tmp = L.build_add (L.build_load counter "counter"
           !builder) (L.const_int i32_t 1) "tmp" !builder in
        ignore(L.build_store tmp counter !builder) in
      ignore(llvm_for function_ptr builder (init_j, predicate_j,
         update_j, body_stmt_j)) in
    ignore(llvm_for function_ptr builder (init_i, predicate_i,
       update_i, body_stmt_i))
  in


(* 2. Statement construction *)
  (* part of code for generating statement, which used both in
     main function and function definition *)
```

```
 let rec build_stmt (fdecl, function_ptr) local_vars builder stmt current_return=

(* Return the value for a variable or formal argument *)

   let rec expr builder e=
    (*expr builder e auxiliaries *)
     let return_aux e t =
       match t with
         A.Matrix ->
           let m = L.build_load (L.build_struct_gep e 0 "m_mat" !builder) "mat_mat" !builder in
           let r = L.build_load (L.build_struct_gep e 1 "m_r" !builder) "r_mat" !builder in
           let c = L.build_load (L.build_struct_gep e 2 "m_c" !builder) "c_mat" !builder in
           let mat = stack_build_mat_init r c function_ptr builder in
           ignore(mat_assign mat (L.const_int i32_t 0) (L.build_sub r (L.const_int i32_t 1) "tmp" !builder)
                             (L.const_int i32_t 0) (L.build_sub c (L.const_int i32_t 1) "tmp" !builder)
                             e (L.const_int i32_t 0) (L.const_int i32_t 0) function_ptr builder);
           ignore(L.build_free m !builder); ignore(L.build_free e !builder);mat
       | _ -> e
     in
     let rec lookup n access=
       match access with
         Access(prev_access, map) ->
           (try (H.find map n, map)
           with Not_found -> lookup n prev_access)
       | Null -> failwith("Semantic error : variable " ^ n ^ " not declared")
     in
     (* convert A.index type to corresponding integral index in a matrix of size r by c *)

     (* for run time dimension check on matrix *)
     let run_time_property_check function_ptr builder err_msg v1 op v2 else_stmt =
```

```
    let predicate builder= op v1 v2 "tmp" !builder in
    let then_stmt builder = ignore(L.build_call printf_func [| string_format_str ; err_msg |] "printf" !builder);
                      ignore(L.build_call abort_func [| |] "" !builder); builder in
    llvm_if function_ptr builder (predicate, then_stmt, else_stmt)
  in
  let run_time_dim_check function_ptr builder v1 op v2 else_stmt =
    run_time_property_check function_ptr builder mat_dim_err_str v1 op v2 else_stmt
  in

  let index_converter d ind r c builder=
    match ind with
      A.Beg -> L.const_int i32_t 0
    | A.End -> (match d with
              "x" -> L.build_sub r (L.const_int i32_t 1) "tmp" !builder
            | "y" -> L.build_sub c (L.const_int i32_t 1) "tmp" !builder
            | _ -> failwith ("Compiler error : index_converter wrong dimension symbol. "))
    | A.ExprInd(e) -> let e' = expr builder e in
                  if (L.string_of_lltype (L.type_of e')) <> "i32" then failwith ("Semantic error : matrix index must be integer.");
                  let else_stmt builder = builder in
                  (match d with
                    "x" -> ignore(run_time_property_check function_ptr builder mat_bound_err_str (L.const_int i32_t 0) (L.build_icmp L.Icmp.Sgt) e' else_stmt);
                          ignore(run_time_property_check function_ptr builder mat_bound_err_str (L.build_sub r (L.const_int i32_t 1) "tmp" !builder) (L.build_icmp L.Icmp.Slt) e' else_stmt);
                  | "y" -> ignore(run_time_property_check function_ptr builder mat_bound_err_str (L.const_int i32_t 0) (L.build_icmp
```

```
                        L.Icmp.Sgt) e' else_stmt);
                            ignore(run_time_property_check
                               function_ptr builder
                               mat_bound_err_str (L.build_sub c
                               (L.const_int i32_t 1) "tmp"
                               !builder) (L.build_icmp
                               L.Icmp.Slt) e' else_stmt);
                     | _ -> failwith ("Compiler error :
                        index_converter wrong dimension symbol.
                        ")); e'
    in


    match e with
      A.IntLit i -> L.const_int i32_t i
    | A.DoubleLit d -> L.const_float double_t d
    | A.StringLit s -> L.build_global_stringptr s
       "system_string" !builder
    | A.BoolLit b -> L.const_int i1_t (if b then 1 else 0)
    | A.MatrixLit (m, (r, c)) -> build_mat_lit (m, (r,c))
       builder(* matrix is represented as arrays of arrays of
       double in LLVM *)
    | A.Noexpr -> L.const_int i32_t 0
    | A.Noassign -> L.const_int i32_t 0
    | A.Id s ->
       let ptr,_ = lookup s local_vars in
       (match (is_matrix ptr) with
         true -> ptr
       | false -> L.build_load ptr s !builder)
    | A.Binop (e1, op, e2) ->
       let exp1 = expr builder e1
       and exp2 = expr builder e2 in
       (match (is_matrix exp1, is_matrix exp2) with
         (false, false)->
           (let typ1 = L.string_of_lltype (L.type_of exp1)
           and typ2 = L.string_of_lltype (L.type_of exp2) in
           (match (typ1, typ2) with
             ("i1", "i1") -> (match op with
                        A.And   -> L.build_and
                      | A.Or    -> L.build_or
                      | A.Equal -> L.build_icmp L.Icmp.Eq
                      | A.Neq   -> L.build_icmp L.Icmp.Ne
```

```
                        | _       -> failwith("Semantic error : wrong operator used on boolean operands.")
                        ) exp1 exp2 "tmp" !builder
           | ("double", "double") | ("i32", "i32") | ("double", "i32") | ("i32", "double") ->
            let build_op_by_type opf opi =
              (match (typ1, typ2) with
               ("double", "double") -> opf
              | ("i32", "i32") -> opi
              | ("double", "i32") ->
                 (fun e1 e2 n bdr -> let e2' = L.build_sitofp e2 double_t n bdr in
                                 opf e1 e2' "tmp" bdr)
              | ("i32", "double") ->
                 (fun e1 e2 n bdr -> let e1' = L.build_sitofp e1 double_t n bdr in
                                 opf e1' e2 "tmp" bdr)
              | _ -> failwith ("Compiler error : numerical operation matching error at build_op_by_type.")
              )
            in
            (match op with
              A.Add   -> build_op_by_type L.build_fadd L.build_add
            | A.Sub   -> build_op_by_type L.build_fsub L.build_sub
            | A.Mult -> build_op_by_type L.build_fmul L.build_mul
            | A.Div   -> build_op_by_type L.build_fdiv L.build_sdiv
            | A.Rmdr -> build_op_by_type L.build_frem L.build_srem
         | A.Equal -> build_op_by_type (L.build_fcmp L.Fcmp.Oeq) (L.build_icmp L.Icmp.Eq)
         | A.Neq  -> build_op_by_type (L.build_fcmp L.Fcmp.One) (L.build_icmp L.Icmp.Ne)
         | A.Less -> build_op_by_type (L.build_fcmp L.Fcmp.Olt) (L.build_icmp L.Icmp.Slt)
         | A.Leq  -> build_op_by_type (L.build_fcmp L.Fcmp.Ole) (L.build_icmp L.Icmp.Sle)
         | A.Greater -> build_op_by_type (L.build_fcmp L.Fcmp.Ogt) (L.build_icmp L.Icmp.Sgt)
         | A.Geq  -> build_op_by_type (L.build_fcmp L.Fcmp.Oge) (L.build_icmp L.Icmp.Sge)
```

```
            | _ -> failwith ("Semantic error : wrong operator
                used on numerical operands.")
        ) exp1 exp2 "tmp" !builder
          | _ -> failwith ("semantic error : invalid numerical
              operation between type " ^ typ1 ^ " and " ^ typ2)))
        (* matrix operation *)
        | (true, false) | (false, true) ->
          let operator =
            (match op with
              A.Add   -> L.build_fadd
            | A.Sub   -> L.build_fsub
            | A.Mult -> L.build_fmul
            | A.Div   -> L.build_fdiv
            | _       -> failwith ("Semantic error : wrong
                operator used on matrix non-matrix operation.")
            )
          in
          (match (is_matrix exp1, is_matrix exp2) with
          | (true, false) -> let typ2 = L.string_of_lltype
              (L.type_of exp2) in
                          (match typ2 with
                            "double" -> mat_num_element_wise exp1
                                exp2 operator function_ptr builder
                          | _ -> failwith("Semantic error :
                              invalid numerical operation between
                              type matrix and "^ typ2))
          | (false, true) -> let typ1 = L.string_of_lltype
              (L.type_of exp1) in
                          (match typ1 with
                            "double" -> mat_num_element_wise exp2
                                exp1 operator function_ptr builder
                          | _ -> failwith("Semantic error :
                              invalid numerical operation between
                              type " ^ typ1 ^ " and matrix."))
          | _ -> failwith("Compiler error : Binop operator
              matching error."))
        | (true, true) ->
          (match op with
            A.Filter -> expr builder (A.Call("filter",[e1; e2]))
          | A.Matprod ->
              let j1 = L.build_load (L.build_struct_gep exp1 2
                  "m_c" !builder) "c_mat" !builder in
```

```
                let i2 = L.build_load (L.build_struct_gep exp2 1 "m_r" !builder) "r_mat" !builder in
                let else_stmt builder= builder in
                ignore(run_time_dim_check function_ptr builder j1 (L.build_icmp L.Icmp.Ne) i2 else_stmt);
                mat_mat_product exp1 exp2 function_ptr builder
            | _ ->
                let i1 = L.build_load (L.build_struct_gep exp1 1 "m_r" !builder) "r_mat" !builder in
                let i2 = L.build_load (L.build_struct_gep exp2 1 "m_r" !builder) "r_mat" !builder in
                let else_stmt builder =
                  let j1 = L.build_load (L.build_struct_gep exp1 2 "m_c" !builder) "c_mat" !builder in
                  let j2 = L.build_load (L.build_struct_gep exp2 2 "m_c" !builder) "c_mat" !builder in
                  let else_stmt builder =
                    builder
                  in
                  run_time_dim_check function_ptr builder j1 (L.build_icmp L.Icmp.Ne) j2 else_stmt
                in
                ignore(run_time_dim_check function_ptr builder i1 (L.build_icmp L.Icmp.Ne) i2 else_stmt);
                (match op with
                      A.Equal -> mat_equal exp1 exp2 function_ptr builder
                    | A.Neq -> mat_not_equal exp1 exp2 function_ptr builder
                    | A.Add  -> mat_mat_element_wise exp1 exp2 L.build_fadd function_ptr builder
                    | A.Sub  -> mat_mat_element_wise exp1 exp2 L.build_fsub function_ptr builder
                    | A.Mult -> mat_mat_element_wise exp1 exp2 L.build_fmul function_ptr builder
                    | A.Div  -> mat_mat_element_wise exp1 exp2 L.build_fdiv function_ptr builder
                    | _      -> failwith ("Semantic error : wrong operator used on matrix operation.")) ))
      | A.Unop(op, e) ->
     let e' = expr builder e in
         let typ = L.string_of_lltype (L.type_of e') in
     (match op with
```

```
    A.Neg ->
          (match typ with
            "i32" -> L.build_neg
          | "double" -> L.build_fneg
          | _ -> failwith ("Semantic error : wrong operands for unary negation operator."))
      | A.Not when typ = "i1"-> L.build_not
      | _ -> failwith ("Semantic error : illegal unary operation.") )e' "tmp" !builder
    | A.Assign (e1, e2) ->
        let single_assign e1 value =
          (match e1 with
            A.Id s ->
              let ptr,map = lookup s local_vars in
              (match (is_matrix ptr) with
                true ->
                  if (L.string_of_lltype (L.type_of value) <> "%matrix_t*")
                  then failwith("Semantic error : matrix must be assigned to a matrix.");
                  let r = L.build_load (L.build_struct_gep value 1 "m_r" !builder) "r_mat" !builder in
                  let c = L.build_load (L.build_struct_gep value 2 "m_c" !builder) "c_mat" !builder in
                  let m = stack_build_mat_init r c function_ptr builder in
                  H.replace map s m;
                  ignore(mat_assign m (L.const_int i32_t 0) (L.build_sub r (L.const_int i32_t 1) "tmp" !builder)
                                  (L.const_int i32_t 0) (L.build_sub c (L.const_int i32_t 1) "tmp" !builder)
                                  value (L.const_int i32_t 0) (L.const_int i32_t 0) function_ptr builder); value
              | false ->
                  let typ1 = L.string_of_lltype (L.type_of (L.build_load ptr "tmp" !builder)) in
                  let typ2 = L.string_of_lltype (L.type_of value) in
                  if (typ1 <> typ2) then failwith ("Semantic error : type "^typ1^" is assigned with type " ^typ2);
                  ignore(L.build_store value ptr !builder); value)
```

```
    | A.Index (s, (A.Range(x_low, x_high), A.Range(y_low, y_high))) ->
      let ptr,_ = lookup s local_vars in
      let r = L.build_load (L.build_struct_gep ptr 1 "m_r" !builder) "r_mat" !builder in
      let c = L.build_load (L.build_struct_gep ptr 2 "m_c" !builder) "c_mat" !builder in
      let x_l = index_converter "x" x_low r c builder in
      let x_h = index_converter "x" x_high r c builder in
      let y_l = index_converter "y" y_low r c builder in
      let y_h = index_converter "y" y_high r c builder in
      if ((x_low = x_high) && (y_low = y_high))
      then (
        if (L.string_of_lltype (L.type_of value)) <> "double" then failwith ("Syntax error : single matrix entry must be assigned with a double");
        let mat = L.build_load (L.build_struct_gep ptr 0 "mat" !builder) "mat" !builder in
        L.build_store value (access mat r c x_l y_l builder) !builder)
      else (
        let i1 = L.build_add (L.build_sub x_h x_l "tmp" !builder) (L.const_int i32_t 1) "tmp" !builder in
        let i2 = L.build_load (L.build_struct_gep value 1 "m_r" !builder) "r_mat" !builder in
        ignore(run_time_property_check function_ptr builder mat_assign_err_str i1 (L.build_icmp L.Icmp.Ne) i2 (fun builder -> builder));
        let j1 = L.build_add (L.build_sub y_h y_l "tmp" !builder) (L.const_int i32_t 1) "tmp" !builder in
        let j2 = L.build_load (L.build_struct_gep value 2 "m_r" !builder) "r_mat" !builder in
        ignore(run_time_property_check function_ptr builder mat_assign_err_str j1 (L.build_icmp L.Icmp.Ne) j2 (fun builder -> builder));
        ignore(mat_assign ptr x_l x_h y_l y_h value (L.const_int i32_t 0) (L.const_int i32_t 0) function_ptr builder); value)
    | _ -> failwith ("Semantic error : only variable and matrix indexing can be assigned to."))
  in
```

```
       let value = expr builder e2 in
       (match e1 with
        A.Comma s_list ->
         (match e2 with
          A.Call(f,_) ->
            let (_, fdecl) = H.find function_decls f in
            let l = match fdecl.A.typ with A.Mulret li -> li | _ -> failwith("Compiler error : Assign expr at A.Call return type is not Mulret") in
            let l1 = List.length s_list in
            let l2 = List.length l in
            if (l1 <> l2) then failwith("Semantic error : "^string_of_int(l1)^" variables are assigned to function call "^f^" which returns "^string_of_int(l2)^" variables.");
            (for i = 0 to ((List.length l) - 1) do
              let v = L.build_load (L.build_struct_gep value i "v_ptr" !builder) "v" !builder in
              ignore (single_assign (List.nth s_list i) (return_aux v (List.nth l i)))
            done);
            ignore(L.build_free value !builder);
         | _ -> failwith("Syntax error: multiple variables must be assigned with a function call that has multiple return values.") ); value
       | _ -> single_assign e1 value
       )

    | A.Index (s, (A.Range(x_low, x_high), A.Range(y_low, y_high))) ->
       let ptr, _ = lookup s local_vars in
       let r = L.build_load (L.build_struct_gep ptr 1 "m_r" !builder) "r_mat" !builder in
       let c = L.build_load (L.build_struct_gep ptr 2 "m_c" !builder) "c_mat" !builder in
       let x_l = index_converter "x" x_low r c builder in
       let x_h = index_converter "x" x_high r c builder in
       let y_l = index_converter "y" y_low r c builder in
       let y_h = index_converter "y" y_high r c builder in
       if ((x_low = x_high) && (y_low = y_high))
       then (
        let mat = L.build_load (L.build_struct_gep ptr 0 "mat" !builder) "mat" !builder in
```

```
          L.build_load (access mat r c x_l y_l builder) "element" !builder)
        else (
          let x_size = L.build_sub x_h x_l "tmp" !builder in
          let y_size = L.build_sub y_h y_l "tmp" !builder in
          let m = stack_build_mat_init (L.build_add x_size (L.const_int i32_t 1) "tmp" !builder)
                              (L.build_add y_size (L.const_int i32_t 1) "tmp" !builder) function_ptr builder in
          ignore(mat_assign m (L.const_int i32_t 0) x_size (L.const_int i32_t 0) y_size ptr x_l y_l function_ptr builder); m)
      (*| A.Index (s, (Range(x_low, x_high), Range(y_low, y_high))) ->
        let (t,ptr) = lookup s in
        let A.Sizedmat(r, c) = t in
        ptr*)
      | A.Call ("printf", [e]) ->
        let exp1 = expr builder e in
        (match (typ_of_lvalue exp1) with
          A.Double -> L.build_call printf_func [| double_format_str ; (exp1) |] "printf" !builder
        | A.Int -> L.build_call printf_func [| int_format_str ; (exp1) |] "printf" !builder
        | A.Bool ->
           let predicate builder = L.build_icmp L.Icmp.Ne (L.const_int i1_t 1) exp1 "tmp" !builder in
           let then_stmt builder = ignore(L.build_call printf_func [| string_format_str ; false_str |] "printf" !builder); builder in
           let else_stmt builder = ignore(L.build_call printf_func [| string_format_str ; true_str |] "printf" !builder); builder in
           ignore(llvm_if function_ptr builder (predicate, then_stmt, else_stmt));
           L.build_call printf_func [| string_format_str ; empty_str |] "printf" !builder
        | A.Matrix -> mat_print exp1 function_ptr builder
        | A.String -> L.build_call printf_func [| string_format_str ; (exp1) |] "printf" !builder
        | _ -> failwith("Compiler error : unknown type expr passed to printf.")
```

```
        )
    | A.Call ("printend", []) ->
        L.build_call printf_func [| string_format_str ; new_line_str |] "printf" !builder
    | A.Call (f, act) ->
      let (fdef, fdecl) =
        match !current_return with
          Maintype | Returnstruct(_) ->
            (try H.find function_decls f with Not_found -> failwith ("Semantic error : function "^f^" not defined."))
        | _ -> H.find function_decls f
      in
      let actuals = List.rev (List.map (expr builder) (List.rev act)) in
      if (List.length actuals) <> (List.length fdecl.A.formals)
      then failwith("Semantic error : expecting " ^ string_of_int (List.length fdecl.A.formals) ^ " arguments in function call "^f);
      List.iter2 (fun (t, _) actual -> if typ_of_lvalue(actual) <> t
                                  then failwith ("Semantic error : wrong type of arguments in function call "^f))
                                  fdecl.A.formals actuals;
      let result =
        (match fdecl.A.typ with
          A.Void -> ""
        | _ -> f ^ "_result")
      in
      let exp = L.build_call fdef (Array.of_list actuals) result !builder in(* corresponding to call void @foo(i32 2, i32 1) *)

      (match fdecl.A.typ with
        A.Void -> exp
      | A.Mulret l ->
          (match (List.length l) with
            1 -> let v = L.build_load (L.build_struct_gep exp 0 "v_ptr" !builder) "v" !builder in
                 ignore(L.build_free exp !builder);
                 return_aux v (List.hd l)
```

```
          | _ -> exp)(* multi return case, can only be used in
              A.Assign, and we will deal with it there. *) (*
              there is a memory leak here due to possible
              multi-return funciton call without assignment,
              haven't got time to tie up *)
        | _ -> failwith ("Compiler error : Call expr function
            return type neither Void nor Mulret."))
      | A.Comma(_) -> failwith("Syntax error : Wrong usage of
          comma seperated list.")
      | _ -> failwith("Syntax error : Wrong usage of matrix
          indexing, possible standalone indexing expressions.")
    in

    match stmt with
    (* Build the code for the given statement; return the builder
        for
       the statement's successor *)
      A.Block sl ->
        let local_vars = Access(local_vars, H.create 1000) in
        let build_st st = ignore (build_stmt (fdecl, function_ptr)
            local_vars builder st current_return) in
        List.iter build_st sl; builder
    | A.Expr e -> ignore (expr builder e); builder
    | A.Return e ->
  (* Since we are infering return type from e, we need to
      consider if a funciton is recursive, and thus when we build
      the function return in its body, its return type has not
      yet been inferred, and its definition is not seen, so it
      cannot call itself because it cannot find itself in
      function_decls, but the thing is that recursive function
      always has a base case (i.e. a return whose return value is
      not recursing on itself, and that we can infer on, so we
      just need to find that return, and use its type as our
      return type *)
        (let eval_return e=
          let e' = expr builder e in
          match (is_matrix e') with
            true -> (* alloca space in heap to temporarily store
                the matrix struct, otherwise the matrix struct is
                stored in the stack of the function that is
                returning, so after return, the stack would be
                cleared, and we might have the matrix just storing
                garbage information. *)
```

```
          let r = L.build_load (L.build_struct_gep e' 1 "m_r" !builder) "r_mat" !builder in
          let c = L.build_load (L.build_struct_gep e' 2 "m_c" !builder) "c_mat" !builder in
          let mat = heap_build_mat_init r c function_ptr builder in
          ignore(mat_assign mat (L.const_int i32_t 0) (L.build_sub r (L.const_int i32_t 1) "tmp" !builder)
                           (L.const_int i32_t 0) (L.build_sub c (L.const_int i32_t 1) "tmp" !builder)
                           e' (L.const_int i32_t 0) (L.const_int i32_t 0) function_ptr builder);mat
      | false -> e'
    in
    let build_return l t=
      let build_return_struct l return=
        for i = 0 to ((List.length l)-1) do
          let e = List.nth l i in
          let e' = eval_return e in
          ignore(L.build_store e' (L.build_struct_gep return i ("return"^string_of_int(i)) !builder) !builder);
        done
      in
      let return = L.build_malloc t "return" !builder in
      (*L.build_store (L.const_int i32_t (List.length l)) (L.build_struct_gep return 0 "return_size" !builder) !builder;*)
      build_return_struct l return;
      ignore(L.build_ret return !builder)
    in
    match !current_return with
      Maintype -> ignore(L.build_ret (L.const_int i32_t 0) !builder)
    | Returnstruct t -> (* this is used to build actual function body *)
        (match e with
          A.Noexpr -> ignore(L.build_ret_void !builder)
        | A.Comma l-> build_return l t;
        | _ -> let l = [e] in build_return l t);
    | Lltypearray ltyp_arr-> (* this is used for return type inference *)
```

```
        (match e with
         A.Noexpr -> current_return := Voidtype(void_t)
        | A.Comma l ->
         (match ltyp_arr with
           [||] -> current_return := Lltypearray(Array.make
              (List.length l) void_t)
         | _ -> ());
         let ltyp_arr = match !current_return with Lltypearray
            l -> l | _ -> failwith("Compiler error :
            Lltypearray wrong matching.") in
         for i = 0 to ((List.length l)-1) do
           try let e'= expr builder (List.nth l i) in
            ltyp_arr.(i) <- L.type_of e';
           with Not_found -> ()
         done
        | _ ->
         (match ltyp_arr with
           [||] -> current_return := Lltypearray([|void_t|])
         | _ -> ());
         let ltyp_arr = match !current_return with Lltypearray
            l -> l | _ -> failwith("Compiler error :
            Lltypearray wrong matching.") in
         try let e'= expr builder e in
          ltyp_arr.(0) <- L.type_of e';
         with Not_found -> ())
      | Voidtype(_) -> ()
      );builder
  | A.If (predicate, then_stmt, else_stmt) ->
    let cond = expr builder predicate in
    if ((L.string_of_lltype (L.type_of cond)) <> "i1") then
       failwith ("Semantic error : predicate of if clause is
       not boolean.");
    let pred builder = expr builder predicate in
    let then_st builder = build_stmt (fdecl, function_ptr)
       local_vars builder then_stmt current_return in
    let else_st builder = build_stmt(fdecl, function_ptr)
       local_vars builder else_stmt current_return in
    ignore(llvm_if function_ptr builder (pred, then_st,
       else_st)); builder

  | A.While (predicate, body) ->
    let cond = expr builder predicate in
```

```
    if ((L.string_of_lltype (L.type_of cond)) <> "i1") then
       failwith ("Semantic error : predicate of while loop is
       not boolean.");
    let pred builder = expr builder predicate in
    let body_st builder = build_stmt (fdecl, function_ptr)
       local_vars builder body current_return in
    ignore(llvm_while function_ptr builder (pred, body_st));
       builder

  | A.For (e1, e2, e3, body) ->
    let cond = expr builder e2 in
    if ((L.string_of_lltype (L.type_of cond)) <> "i1") then
       failwith ("Semantic error : predicate of for loop is not
       boolean.");
    let init_st builder = expr builder e1 in
    let pred builder = expr builder e2 in
    let update builder = ignore(expr builder e3); builder in
    let body_st builder = build_stmt (fdecl, function_ptr)
       local_vars builder body current_return in
    ignore(llvm_for function_ptr builder (init_st, pred,
       update, body_st)); builder
  | A.Local (t, n, v) -> let map = match local_vars with
     Access(_, map) -> map | Null -> failwith("Compiler error :
     local access link error") in
                  (match t with
                   A.Matrix ->
                     (match v with
                      A.Noassign -> let local =
                          stack_build_mat_init (L.const_int
                          i32_t 0) (L.const_int i32_t 0)
                          function_ptr builder in
                                 H.add map n local;
                     | _-> let v' = expr builder v in
                          if ((L.string_of_lltype (L.type_of
                             v')) <> "%matrix_t*") then
                             failwith ("Semantic error : Right
                             hand side of the matrix
                             definition of "^n^" is not a
                             matrix expression");
                          let r = L.build_load
                             (L.build_struct_gep v' 1 "m_r"
                             !builder) "r_mat" !builder in
```

```
                    let c = L.build_load
                        (L.build_struct_gep v' 2 "m_c"
                        !builder) "c_mat" !builder in
                    let local = stack_build_mat_init r c
                        function_ptr builder in
                    ignore(mat_assign local (L.const_int
                        i32_t 0) (L.build_sub r
                        (L.const_int i32_t 1) "tmp"
                        !builder)
                             (L.const_int i32_t 0)
                                (L.build_sub c
                                (L.const_int i32_t 1)
                                "tmp" !builder)
                             v' (L.const_int i32_t 0)
                                (L.const_int i32_t 0)
                                function_ptr builder);
                    H.add map n local)
           | _ ->
               let local = L.build_alloca (ltype_of_typ
                   t) n !builder in
               H.add map n local;
               let init_v =
                 (match v with
                   A.Noassign ->
                     (match t with
                       A.Int -> L.const_int i32_t 0
                     | A.Double -> L.const_float double_t
                         0.
                     | A.String ->
                         L.build_global_stringptr ""
                         "system_string" !builder (*empty
                         string*)
                     | A.Bool -> L.const_int i1_t 0
                     | _ -> failwith ("Compiler error :
                         local variable type matching
                         error."))
                 | _ -> expr builder v)
               in
               let typ = L.string_of_lltype (L.type_of
                   (L.build_load local "tmp" !builder))
                   in
               if ((L.string_of_lltype (L.type_of
                   init_v)) <> typ) then failwith
```

```
                                    ("Semantic error : Right hand side of
                                    the definition of "^n^" is not type "
                                    ^ typ);
                              ignore(L.build_store init_v local
                                    !builder)
                         );builder
  in



(* 3. User-defined function *)
  (* Fill in the body of the given function *)
  let build_function_body fdecl =
    let current_return = ref (Lltypearray([||])) in (* will be
        used to stored the lltype of last return expression
        encountered in a function body*)
    let formal_types =
      let f(t,_) =
        match t with
          A.Matrix -> L.pointer_type matrix_t
        | _ -> ltype_of_typ t
      in
      Array.of_list (List.map f fdecl.A.formals) in
     (* User-defined function body construction *)
    let body_building function_ptr =
      let builder = ref (L.builder_at_end context (L.entry_block
          function_ptr)) in
      (* imagine entry_block returns a block (i.e. {block} ), and
          builder_at_end enables adding instructions at the end of
          the block??*)
      (* Construct the function's "locals": formal arguments and
          locally
       declared variables. Allocate each on the stack, initialize
           their
       value, if appropriate, and remember their values in the
           "locals" map *)
      let local_vars =
        let add_formal m (t, n) p = (* L.set_value_name n p; *)(*
            p is a value not a ptr? *) (*!! set_value_name returns
            () *)
          match t with
            A.Matrix ->
```

```
          let r = L.build_load (L.build_struct_gep p 1 "m_r" !builder) "r_mat" !builder in
          let c = L.build_load (L.build_struct_gep p 2 "m_c" !builder) "c_mat" !builder in
          let local = stack_build_mat_init r c function_ptr builder in
          ignore(mat_assign local (L.const_int i32_t 0) (L.build_sub r (L.const_int i32_t 1) "tmp" !builder)
                      (L.const_int i32_t 0) (L.build_sub c (L.const_int i32_t 1) "tmp" !builder)
                      p (L.const_int i32_t 0) (L.const_int i32_t 0) function_ptr builder);
          H.add m n local;m
        | _ ->
         let local = L.build_alloca (ltype_of_typ t) n !builder in
         ignore (L.build_store p local !builder);
         H.add m n local;m(* local is a ptr? *)
      in
      let func_local_access = Access(Null, H.create (1000 + List.length fdecl.A.formals)) in
      let map = match func_local_access with Access(_, map) -> map | Null -> failwith("Compiler error : function local access link error") in
      ignore(List.fold_left2 add_formal map fdecl.A.formals (Array.to_list (L.params function_ptr))); func_local_access
    in
    (* Build the code for each statement in the function *)
    builder := !(build_stmt (fdecl, function_ptr) local_vars builder (A.Block fdecl.A.body) current_return);
    match !current_return with
      Returnstruct t ->
        add_terminal builder (match fdecl.A.typ with
          A.Void -> L.build_ret_void
        | _ -> L.build_ret (L.build_alloca t "tmp" !builder))
    | _ -> () (* this is when doing type inference, the system function is going to be deleted anyway, so we don't care if all its blocks have ret or not *)
  in
  (* temporary function to go through code once, so that we can do return type inference *)
```

```
    let system_function = L.define_function "system_function" (L.function_type void_t formal_types) the_module in
    body_building system_function;
    (* find return type from current_return *)
    let return_t = L.named_struct_type context "return_t" in
    (match !current_return with
      Voidtype(_) -> current_return := Returnstruct (void_t); ignore(fdecl.A.typ <- A.Void)
    | Lltypearray ltyp_arr -> L.struct_set_body return_t ltyp_arr false; current_return := Returnstruct (return_t);
                        let f m t = (type_of_lltype t)::m in
                        ignore(fdecl.A.typ <- A.Mulret (List.rev (Array.fold_left f [] ltyp_arr)))
    | Returnstruct(_) -> failwith ("Compiler error : type inference bug")
    | Maintype -> failwith ("Compiler error : type inference bug") );
    (* User-defined function declarations *)
    let name = fdecl.A.fname in
    let return_type =
      let ret = match !current_return with Returnstruct t -> t | _-> failwith ("Compiler error : type inference bug") in
      match (L.string_of_lltype ret) with
        "void" -> void_t
      | _ -> L.pointer_type ret
    in
    let ftype = L.function_type return_type formal_types in
    let function_decl = L.define_function name ftype the_module in
    H.add function_decls name (function_decl, fdecl);
    body_building function_decl;
    L.delete_function system_function (*for some unknown reason, it seems that deleting this auxiliary function would trigger stack protector and segment fault, so we have to let it be *)
  in


(* 4. Main function body construction *)

  (* build main function *)
  let build_main main_body =
    let current_return = ref Maintype in

```

```
 (* continue with building main function *)
   let main_fdecl = {
     A.typ = A.Int;
     A.fname = main_name;
     A.formals = [];
     A.body = main_body;
     } in


   let local_vars = Access(Null, H.create 1000) in
   main_builder := !(build_stmt (main_fdecl, main_define)
       local_vars main_builder (A.Block main_fdecl.A.body)
       current_return);
    (* Add a return if the last block falls off the end *)

   add_terminal main_builder (L.build_ret (L.const_int i32_t 0))
       in


(* 5. Combine all *)

 (* built-in functions *)
  let built_in_body_building f body=
   let (fdef, _) = H.find function_decls f in
   body fdef
  in

 (* i. size() *)
(* define size(), which return matrix size *)
  let size_func_decl =
   { A.typ = A.Mulret([A.Int; A.Int]);
     A.fname = "size";
     A.formals = [(A.Matrix, "mat")];
     A.body = [] }
  in
  let matrix_size_t = L.named_struct_type context
      "matrix_size_t" in
  L.struct_set_body matrix_size_t [|i32_t; i32_t|] false;
  let size_func =
   L.define_function "size" (L.function_type (L.pointer_type
       matrix_size_t) [| L.pointer_type matrix_t |]) the_module
  in
```

```
  H.add function_decls "size" (size_func, size_func_decl);
   (* size function body *)
  let size_func_body function_ptr =
    let builder = ref (L.builder_at_end context (L.entry_block function_ptr)) in
    let return = L.build_malloc matrix_size_t "return" !builder in
    let p = List.hd (Array.to_list (L.params function_ptr)) in
    let r = L.build_load (L.build_struct_gep p 1 "m_r" !builder) "r_mat" !builder in
    ignore(L.build_store r (L.build_struct_gep return 0 "row_size" !builder) !builder);
    let c = L.build_load (L.build_struct_gep p 2 "m_c" !builder) "c_mat" !builder in
    ignore(L.build_store c (L.build_struct_gep return 1 "col_size" !builder) !builder);
    ignore(L.build_ret return !builder)
  in
  built_in_body_building "size" size_func_body;

 (* ii. zeros(i,j) *)
(* define zeros(i,j), which return a zero matrix *)
  let zero_matrix_func_decl =
    { A.typ = A.Mulret([A.Matrix]);
      A.fname = "zeros";
      A.formals = [(A.Int, "i"); (A.Int, "j")];
      A.body = [] }
  in
  let zero_matrix_t = L.named_struct_type context "zero_matrix_t" in
  L.struct_set_body zero_matrix_t [| L.pointer_type matrix_t |] false;
  let zero_matrix_func =
    L.define_function "zeros" (L.function_type (L.pointer_type zero_matrix_t) [| i32_t; i32_t |]) the_module
  in
  H.add function_decls "zeros" (zero_matrix_func, zero_matrix_func_decl);
   (* zeros function body *)
  let zero_matrix_func_body function_ptr =
    let builder = ref (L.builder_at_end context (L.entry_block function_ptr)) in
    let return = L.build_malloc zero_matrix_t "return" !builder in
    let i = List.hd (Array.to_list (L.params function_ptr)) in
```

```
    let j = List.hd (List.tl (Array.to_list (L.params function_ptr))) in
    let m = heap_build_mat_init i j function_ptr builder in
    ignore(L.build_store m (L.build_struct_gep return 0 "m" !builder) !builder);
    ignore(L.build_ret return !builder)
  in
  built_in_body_building "zeros" zero_matrix_func_body;

  (*iii. int2double(i) *)
 let int_to_double_func_decl =
    { A.typ = A.Mulret([A.Double]);
      A.fname = "int2double";
      A.formals = [(A.Int, "i")];
      A.body = [] }
  in
  let int_to_double_t = L.named_struct_type context "int_to_double_t" in
  L.struct_set_body int_to_double_t [| double_t |] false;
  let int_to_double_func =
    L.define_function "int2double" (L.function_type (L.pointer_type int_to_double_t) [| i32_t |]) the_module
  in
  H.add function_decls "int2double" (int_to_double_func, int_to_double_func_decl);
   (* int2double function body *)
  let int_to_double_func_body function_ptr =
    let builder = ref (L.builder_at_end context (L.entry_block function_ptr)) in
    let return = L.build_malloc int_to_double_t "return" !builder in
    let i = List.hd (Array.to_list (L.params function_ptr)) in
    let d = L.build_sitofp i double_t "tmp" !builder in
    ignore(L.build_store d (L.build_struct_gep return 0 "converted_double" !builder) !builder);
    ignore(L.build_ret return !builder)
  in
  built_in_body_building "int2double" int_to_double_func_body;

(*iv. double2int(d) *)
 let double_to_int_func_decl =
    { A.typ = A.Mulret([A.Int]);
      A.fname = "double2int";
```

```
    A.formals = [(A.Double, "d")];
    A.body = [] }
  in
  let double_to_int_t = L.named_struct_type context "double_to_int_t" in
  L.struct_set_body double_to_int_t [| i32_t |] false;
  let double_to_int_func =
   L.define_function "double2int" (L.function_type (L.pointer_type double_to_int_t) [| double_t |]) the_module
  in
  H.add function_decls "double2int" (double_to_int_func, double_to_int_func_decl);
   (* double2int function body *)
  let double_to_int_func_body function_ptr =
   let builder = ref (L.builder_at_end context (L.entry_block function_ptr)) in
   let return = L.build_malloc double_to_int_t "return" !builder in
   let d = List.hd (Array.to_list (L.params function_ptr)) in
   let i = L.build_fptosi d i32_t "tmp" !builder in
   ignore(L.build_store i (L.build_struct_gep return 0 "converted_int" !builder) !builder);
   ignore(L.build_ret return !builder)
  in
  built_in_body_building "double2int" double_to_int_func_body;

 (*v. load(filename) *)
  let load_cpp_t = L.function_type (L.pointer_type double_t) [| str_t |] in
  let load_cpp_func = L.declare_function "load_cpp" load_cpp_t the_module in
  let load_func_decl =
   { A.typ = A.Mulret([A.Matrix; A.Matrix; A.Matrix]);
    A.fname = "load";
    A.formals = [(A.String, "filename")];
    A.body = [] }
  in
  let load_t = L.named_struct_type context "load_t" in
  L.struct_set_body load_t [| L.pointer_type matrix_t ; L.pointer_type matrix_t ; L.pointer_type matrix_t |] false;
  let load_func =
   L.define_function "load" (L.function_type (L.pointer_type load_t) [| str_t |]) the_module
```

```
 in
 H.add function_decls "load" (load_func, load_func_decl);
 let load_func_body function_ptr =
  let builder = ref (L.builder_at_end context (L.entry_block function_ptr)) in
  let return = L.build_malloc load_t "return" !builder in
  let path = List.hd (Array.to_list (L.params function_ptr)) in
  let mat_arr = L.build_call load_cpp_func [| path |] "mat_arr" !builder in
  let i = L.build_fptosi (L.build_load (L.build_gep mat_arr [|L.const_int i32_t 0|] "element_ptr" !builder) "tmp" !builder) i32_t "tmp" !builder in
  let j = L.build_fptosi (L.build_load (L.build_gep mat_arr [|L.const_int i32_t 1|] "element_ptr" !builder) "tmp" !builder) i32_t "tmp" !builder in
  let return_mat_r = heap_build_mat_init i j function_ptr builder in
  let return_mat_g = heap_build_mat_init i j function_ptr builder in
  let return_mat_b = heap_build_mat_init i j function_ptr builder in
  to_rgb_matrix mat_arr return_mat_r return_mat_g return_mat_b i j function_ptr builder;
  ignore(L.build_store return_mat_r (L.build_struct_gep return 0 "mat_r" !builder) !builder);
  ignore(L.build_store return_mat_g (L.build_struct_gep return 1 "mat_r" !builder) !builder);
  ignore(L.build_store return_mat_b (L.build_struct_gep return 2 "mat_r" !builder) !builder);
  ignore(L.build_ret return !builder)
 in
 built_in_body_building "load" load_func_body;

(*vi. save(mat_r, mat_g, mat_b, filename) *)
 let save_cpp_t = L.function_type void_t [| L.pointer_type double_t; str_t |] in
 let save_cpp_func = L.declare_function "save_cpp" save_cpp_t the_module in
 let save_func_decl =
  { A.typ = A.Void;
   A.fname = "save";
   A.formals = [(A.Matrix, "r");(A.Matrix, "g");(A.Matrix, "b");(A.String, "filename")];
```

```
      A.body = [] }
  in
  let save_func =
    L.define_function "save" (L.function_type void_t [|
        L.pointer_type matrix_t; L.pointer_type matrix_t;
        L.pointer_type matrix_t; str_t |]) the_module
  in
  H.add function_decls "save" (save_func, save_func_decl);
  let save_func_body function_ptr =
    let builder = ref (L.builder_at_end context (L.entry_block
        function_ptr)) in
    let act = Array.to_list (L.params function_ptr) in
    let m_r = List.nth act 0 in
    let m_g = List.nth act 1 in
    let m_b = List.nth act 2 in
    let path = List.nth act 3 in
    let i = L.build_load (L.build_struct_gep m_r 1 "m_r"
        !builder) "r_mat" !builder in
    let j = L.build_load (L.build_struct_gep m_r 2 "m_c"
        !builder) "c_mat" !builder in
    let size = L.build_add (L.build_mul (L.build_mul i j "tmp"
        !builder) (L.const_int i32_t 3) "tmp" !builder)
        (L.const_int i32_t 2) "tmp" !builder in
    let return_arr = L.build_array_malloc double_t size
        "return_arr" !builder in
    from_rgb_matrix return_arr m_r m_g m_b i j function_ptr
        builder;
    ignore(L.build_call save_cpp_func [| return_arr; path |] ""
        !builder);
    ignore(L.build_ret_void !builder)
  in
  built_in_body_building "save" save_func_body;


(*vii. face(filename) *)
  let faceDetect_t = L.function_type (L.pointer_type double_t)
      [| str_t |] in
  let faceDetect_func = L.declare_function "faceDetect"
      faceDetect_t the_module in
  let face_func_decl =
    { A.typ = A.Mulret([A.Matrix]);
      A.fname = "face";
      A.formals = [(A.String, "filename")];
```

```
      A.body = [] }
  in
  let face_t = L.named_struct_type context "face_t" in
  L.struct_set_body face_t [| L.pointer_type matrix_t|] false;
  let face_func =
   L.define_function "face" (L.function_type (L.pointer_type
      face_t) [| str_t |]) the_module
  in
  H.add function_decls "face" (face_func, face_func_decl);
  let face_func_body function_ptr =
   let builder = ref (L.builder_at_end context (L.entry_block
      function_ptr)) in
   let return = L.build_malloc face_t "return" !builder in
   let path = List.hd (Array.to_list (L.params function_ptr)) in
   let mat_arr = L.build_call faceDetect_func [| path |]
      "mat_arr" !builder in
   let num = L.build_fptosi (L.build_load (L.build_gep mat_arr
      [|L.const_int i32_t 0|] "element_ptr" !builder) "tmp"
      !builder) i32_t "tmp" !builder in
   let return_mat_r = heap_build_mat_init (L.const_int i32_t 4)
      num function_ptr builder in
   face_matrix mat_arr return_mat_r num function_ptr builder;
   ignore(L.build_store return_mat_r (L.build_struct_gep return
      0 "mat_r" !builder) !builder);
   ignore(L.build_ret return !builder)
  in
  built_in_body_building "face" face_func_body;

  List.iter build_function_body functions; build_main main_stmt;
  the_module
```

8.7 standard library

```


func bitwise(matrix m, matrix n) {
  double k =0.0;
  int i = 0;
  int j = 0;
  for (i = 0; i<3; i=i+1){
    for (j = 0; j<3; j=j+1){
```

```
      k = k + m[i,j]*n[i,j];
    }
  }
  return k;
}



func filter(matrix m, matrix n) {
  int a;
  int b;
  int c;
  int d;
  a,b = size(m);
  c,d = size(n);
  if (c == 3) {
  matrix t = zeros(a+2,b+2);
  matrix r = zeros(a,b);
  t[1:a,1:b] = m[0:a-1,0:b-1];
  int i = 0;
  int j = 0;
  for (i = 0; i<a; i=i+1){
    for (j = 0; j<b; j=j+1){
     double k = 0.0;
     k = bitwise(t[i:i+2,j:j+2],n);
     r[i,j] = k;
    }
  }
  return r;
  }
  if (c == 5) {
  matrix t = zeros(a+4,b+4);
  matrix r = zeros(a,b);
  t[2:a+1,2:b+1] = m[0:a-1,0:b-1];
  int i = 0;
  int j = 0;
  for (i = 0; i<a; i=i+1){
    for (j = 0; j<b; j=j+1){
     double k = 0.0;
     k = bitwise(t[i:i+4,j:j+4],n);
     r[i,j] = k;
    }
  }
```

```
  return r;
  }
}
```

8.8 ext.cpp (opencv functions)

```
#include <opencv2/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/opencv.hpp>
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

#include <stdio.h>
#include <iostream>
#include <string>

using namespace cv;

using namespace std;
extern "C" double* load_cpp(char imageName[])
{
   Mat img = imread(imageName,CV_LOAD_IMAGE_COLOR);
   unsigned char* input = (unsigned char*)(img.data);
   double* output = new double[2+3*img.rows*img.cols];
   output[0]=img.rows;
   output[1]=img.cols;
   double r,g,b;
   int k = 2;
   for(int i = 0;i < img.rows;i++){
      for(int j = 0;j < img.cols;j++){
         b = input[img.step * i + j*img.channels()] ;
         output[k++]=b;
         g = input[img.step * i + j*img.channels() + 1];
         output[k++]=g;
         r = input[img.step * i + j*img.channels() + 2];
         output[k++]=r;
      }
   }
   return output;
```

```
}

extern "C" void save_cpp(double* input, char fileName[])
{
   int height = input[0];
   int width = input[1];
   double* data = new double[3*width*height];
   for(int i = 0; i < 3*width*height; i++) data[i]=input[i+2];
   Mat image = cv::Mat(height, width, CV_64FC3, data);
   imwrite(fileName,image);
   return;
}

extern "C" double* faceDetect(char fileName[])
{
   Mat image = imread(fileName, CV_LOAD_IMAGE_COLOR);

   // Load Face cascade (.xml file)
   CascadeClassifier face_cascade;
   face_cascade.load(
      "/usr/local/Cellar/opencv/3.3.1_1/share/OpenCV/haarcascades/haarcascade_
      );
    face_cascade.load(
      "/opt/opencv/data/haarcascades/haarcascade_frontalface_alt2.xml"
      );//ubuntu

   // Detect faces
   std::vector<Rect> faces;
   face_cascade.detectMultiScale( image, faces, 1.1, 2,
      0|CV_HAAR_SCALE_IMAGE, Size(30, 30) );

   double* output = new double[1+4*faces.size()];
   output[0]=faces.size();//number of faces
   for( int i = 0; i < faces.size(); i++ )
   {
      output[4*i+1]=faces[i].y + faces[i].height*0.5;
      output[4*i+2]=faces[i].x + faces[i].width*0.5;
      output[4*i+3]=faces[i].height;
      output[4*i+4]=faces[i].width;
      // Point center( faces[i].x + faces[i].width*0.5,
         faces[i].y + faces[i].height*0.5 );
      // ellipse( image, center, Size( faces[i].width*0.5,
         faces[i].height*0.5), 0, 0, 360, Scalar( 255, 0, 255 ),
```

```
        4, 8, 0 );
    }
    return output;
}
```

8.9 compile (shell script for calling Facelab compiler to generate .exe)

```
#!/bin/bash
for var in "$@"
do
    rm $var.ir;
    ./facelab.native $var.fb >> $var.ir;
    llc-5.0 $var.ir;
    clang++-4.0 `pkg-config --cflags opencv` `pkg-config --libs
        opencv` $var.ir.s ext.cpp -o $var
done
```

Below are test cases:

8.10 add1.fb

```
func try() {
   int i = 3;
   int j = 5;
   return i + j;
}
int d;
d = try();
printf(d);
```

8.11 addDouble.fb

```
func addDouble() {
   double i = 3;
   double j = 5;
   return i + j;
}
```

```
double d;
d = addDouble();
printf(d);
```

8.12 conv.fb

```
func bitwise(matrix m, matrix n) {
  double k =0.0;
  int i = 0;
  int j = 0;
  for (i = 0; i<3; i=i+1){
    for (j = 0; j<3; j=j+1){
     k = k + m[i,j]*n[i,j];
    }
  }
  printf(k);
  return k;
}

matrix m = [1.0, 2.0, 3.0;
       4.0, 5.0, 6.0;
       7.0, 8.0, 9.0];

matrix s = [0.0, -1.0, 0.0;
      -1.0, 5.0, -1.0;
      0.0, -1.0, 0.0];


func Filter(matrix m, matrix n) {
  matrix r = [0.0,0.0,0.0;
         0.0,0.0,0.0;
         0.0,0.0,0.0];
  matrix t = [0.0,0.0,0.0,0.0,0.0;
         0.0,0.0,0.0,0.0,0.0;
         0.0,0.0,0.0,0.0,0.0;
         0.0,0.0,0.0,0.0,0.0;
         0.0,0.0,0.0,0.0,0.0];

  r[0:2,0:2] = m[0:2,0:2];
  int i = 0;
  int j = 0;
```

```
  return 0;
  /*for (i = 0; i<3; i=i+1){
    for (j = 0; j<3; j=j+1){
     double k = 0.0;
     k = bitwise(t[i:i+2,j:j+2],n);
     printf(k);
    }
  }*/
}
```

8.13 conv2.fb

```

matrix m = [1.0, 2.0, 3.0, 4.0;
         4.0, 5.0, 6.0,5.0;
     7.0, 8.0, 9.0,6.0];

func f(){
  printf(1);
  printend();
  //printf(m);
  return;
}
f();


matrix s = [0.0, -1.0, 0.0;
      -1.0, 5.0, -1.0;
      0.0, -1.0, 0.0];

matrix t = [0.0, -1.0, 0.0,1.0,1.0;
      -1.0, 5.0, -1.0,1.0,1.0;
      0.0, -1.0, 0.0,1.0,1.0;
      0.0, -1.0, 0.0,1.0,0.0;
      0.0, -1.0, 0.0,0.0,1.0];

matrix r = m $ s $ t;
printf(r);
```

8.14 double2int.fb

```
matrix m = zeros(3,3);
int i; int j;
for (i = 0; i != 3; i = i+1)
{
   for (j=0; j!= 3; j = j+1)
   {
      m[i,j] = i*3+j+(i*3+j)/10.0;
   }
}
printf(m);printend();
for (i = 0; i != 3; i = i+1)
{
   for (j=0; j!= 3; j = j+1)
   {
      printf(double2int(m[i,j]));printf(" ");
   }
}
```

8.15 factorial.fb

```
func factorial (int i)
{
   if (i==1)
   {
      return 1;
   }
   else
   {
      return i * factorial (i-1);
   }
}

printf(factorial(7));printend();
```

8.16 gcd.fb

```
func gcd(int m, int n) {
//calculate gcd of two integer number
while(m!=0 && n!=0)
{
if(n > m) n = n % m;
else m = m % n;
}
if(m ==0) return n;
else return m;
}

int m = gcd(81,18);
printf(m);
```

8.17 gcd_recursive.fb

```
func gcd(int m, int n)
{
   if (m == 0)
      return n;
   if (n == 0)
      return m;
   if (m > n)
      return gcd(m%n, n);
   else
      return gcd(n%m, m);
}
printf(gcd(252, 9)); printend();
printf(gcd(71, 131)); printend();
```

8.18 int2double.fb

```
matrix m = zeros(3,3);
int i; int j;
for (i = 0; i != 3; i = i+1)
{
   for (j=0; j!= 3; j = j+1)
   {
```

```
      m[i,j] = i*3+j+int2double(i*3+j)/10;
   }
}
printf(m);printend();
```

8.19 load_1.fb

```
matrix r; matrix g; matrix b;
r,g,b = load("load_1.jpg");
int i; int j;
i,j = size(r);
printf(i);
printf(r);printend();
printf(g);printend();
printf(b);printend();
```

8.20 load_2.fb

```
matrix r; matrix g; matrix b;
r,g,b = load("load_1.jpg");
int i; int j;
i,j = size(r);
printf(j/2);
//save(r[:i/2, :j/2] ,g[:i/2, 0:j/2] ,b[:i/2, 0:j/2]
   ,"load_2_result.jpg");
printf(r[1:,2:]);
save(r[1:, 2:] ,g[1:, 2:] ,b[1:, 2:] ,"load_2_result.jpg");
```

8.21 main_6.fb

```
func f1() {printf(1); return 5;}
func f2() { string st; printf(f1()); st = "abc";return st;}
int i=2;
/*int j=3;
printf(i); printf(j);
i = 0;
```

```
printf(i);
j = f1();
printf("now j is :");
printf(j);
string my_str;
my_str = "hahaha";
printf(my_str);
string s;
s = f2();
printf("now s is :");
printf(s);*/
```

8.22 main_7.fb

```
int i = 3;
int j = 3+4;
int k = i+j+2;
printf(k);
```

8.23 main_8.fb

```
if(true) printf(1);
int i = 0;
while (i != 3)
{
   printf(i);
   i = i+1;
}
for (i = 0; i!= 3; i=i+1)
{
   printf(i);
}
```

8.24 main_9.fb

```
func f1() {printf(1); return 5;}
```

```
func f2() { string st; printf(f1()); st = "abc";return st;}
func f3(matrix m, double d) {printf("testing");printend();
    return m*d;}
int i=2;
int j=3;
printf(i); printf(j);
i = 0;
printf(i);
j = f1();
printf("now j is :");
printf(j);
string my_str;
my_str = "hahaha";
printf(my_str);
string s;
s = f2();
printf("now s is :");
printf(s);
matrix m = [1.1, 2.2, 3.3; 4.4, 5.5, 6.6];
printf(f3(m,10.01));
```

8.25 matrix_1.fb

```
[1.1,2.2,3.3; 4.4,5.5,6.6];
```

8.26 matrix_2.fb

```
matrix m = [1.0,2.0,3.0;4.0,5.0,6.0;7.0,8.0,9.0;10.0,11.0,12.0];
m;
printf(m);
//printf([1.0,2.0,3.0;4.0,5.0,6.0;7.0,8.0,9.0;10.0,11.0,12.0]);
```

8.27 matrix_3.fb

```
func f(matrix m) { printf(m); return;}
printf("var");
```

```
matrix m = [1.0,2.0,3.0;4.0,5.0,6.0;7.0,8.0,9.0;10.0,11.0,12.0];
m;
printf(m);
printf("fun");
f(m);
printf("lit");
printf([1.0,2.0,3.0;4.0,5.0,6.0;7.0,8.0,9.0;10.0,11.0,12.0]);
```

8.28 matrix_4.fb

```
func f(matrix m) { printf(m); return;}
printf("var");
matrix m = [1.0,2.0,3.0;4.0,5.0,6.0;7.0,8.0,9.0;10.0,11.0,12.0];
m;
printf(m[0:1,0:1]);
printf("fun");
f(m[:, 2:]);
printf("lit");
printf([1.0,2.0,3.0;4.0,5.0,6.0;7.0,8.0,9.0;10.0,11.0,12.0]);
matrix m2 = m[:,:];
printf("fun2:");
f(m2);
matrix m3 = m2[:1,2:];
printf("fun3:");
f(m3);
```

8.29 matrix_5.fb

```
func f(matrix m) { printf(m); return;}
printf("var");
matrix m = [1.0,2.0,3.0;4.0,5.0,6.0;7.0,8.0,9.0;10.0,11.0,12.0];
f(m + 3.0);
f(m * 2.0);
matrix m2 = m / 1.5;
f(m * m2);
```

8.30 matrix_6.fb

```
func f(matrix m) { printf(m); return;}
printf("var");printend();
matrix m = [1.0,2.0,3.0;4.0,5.0,6.0;7.0,8.0,9.0;10.0,11.0,12.0];
printf(m);
printf("fun");printend();
f(3.0 * m - 5.0 * m);
printf("fun2"); printend();
f(m .* [2.2, 4.4; 6.6, 1.5; 9.1, 3.5]);
```

8.31 matrix_7.fb

```
func f(matrix m) { printf(m); return;}
matrix m = [1.0,2.0,3.0;4.0,5.0,6.0;7.0,8.0,9.0;10.0,11.0,12.0];
printf(m == m);
printf(m != m);
printend();
matrix m2 = m / 1.5;
printf(m2 != m);
printf(m2 == m);
printend();
matrix m3 = m / 1.0;
printf(m3 == m);
printf(m3 != m);
printend();
matrix m4 = m * 1.001;
printf(m4 != m);
printf(m4 == m);
printend();
matrix m5 = 0.0 + m;
printf(m5 == m);
printf(m5 != m);
printend();
```

8.32 matrix_9.fb

```
func multiply(matrix m) {
```

```
  matrix m2 = [0.0, 0.1; 1.0, 1.1; 2.0, 2.1; 3.0, 3.1];
  printf(m2[1:,:]);
  matrix m3 = m2[1:,:];
  printend();
  printf(m .* m3);
}
matrix m = [0.0, 0.1, 0.2; 1.0, 1.1, 1.2; 2.0, 2.1, 2.2];
printf(m);
printend();
multiply(m);
```

8.33 matrix_11.fb

```
func funky() {
  matrix m = [0.0,-0.1,0.2;0.0,0.1,0.2;1.1,1.2,1.3];
  printf(m[0,1]);
}
funky();

// can't have negative values in matrix;
// sample output: [-0.1]
```

8.34 matrix_13.fb

```
int i; int j;
matrix m = [1.0,2.0,3.0;4.0,5.0,6.0;7.0,8.0,9.0;10.0,11.0,12.0];
i,j = size(m);
printf(i);printend();printf(j);
```

8.35 matrix_14.fb

```
matrix m = [1.1,2.2;3.3,4.4];
printf(m[0,0]);printend();
printf(m[1,1]);printend();
m[1,0] = 0.0; m[0,1] = 0.0;
printf(m);
```

8.36 matrix_15.fb

```
matrix m = zeros(3,4);
m[2,3] = 2.2;
m[0,0] = 3.3;
printf(m);
```

8.37 multi_ret1.fb

```
func f(matrix m) { printf(m); return;}
func f2(matrix m1, matrix m2, double d) {printf(m1 .* m2 * d);
    return m1*d, m2;}
matrix m = [1.0,2.0,3.0;4.0,5.0,6.0;7.0,8.0,9.0;10.0,11.0,12.0];
printf("fun1"); printend();
f(m .* [2.2, 4.4; 6.6, 1.5; 9.1, 3.5]);
matrix m2 = [2.2, 4.4;6.6, 1.5; 9.1, 3.5];
matrix m3; matrix m4;
m3,m4 = f2(m, m2, 10.0);
printend();f(m3.*m4);
printend();f2(m3,m4,1.0);
```

8.38 multi_ret2.fb

```
func f(string name, matrix m1, matrix m2, matrix m3, double d)
{
   printf(name);printf(" : ");
   printend();
   printf((m1+m2).*m3*d*5.0);
   printend();
   return m1.*m3, m2.*m3, d*5.0;
}
func f2(matrix m1, matrix m2, double d)
{
   matrix m3; matrix m4; double d2;
   m3, m4 ,d2 = f("m1", m1, m1, m2, d);
   printend();
   printf((m3+m4)*d2);
   printend();
```

```
    printf(m3*2.0*d2);
    return 1, 2.0, "haha";
}
int i; double d; string s;
i,d,s = f2([1.0,2.0;3.0,4.0], [8.2,163.4;924.6,99.9], 4.0);
f("m2", [1.0,2.0;3.0,4.0], [1.0,2.0;3.0,4.0],
    [8.2,163.4;924.6,99.9], 4.0);
printf(i); printend(); printf(d); printend();
    printf(s);printend();
printf([1.0,2.0;3.0,4.0]== [1.0,2.0;3.0,4.0]);printend();
matrix m = [8.2,163.4;924.6,99.9];
printf(m == [1.0,2.0;3.0,4.0]);printend();
```

8.39 printdouble.fb

```
double d = 3.0;
printf(d);
```

8.40 printdouble2.fb

```
double d = 3.1;
int i = 2;
double j;
j = i * d;
printf(j);
```

8.41 save_1.fb

```
matrix r = [0.0, 255.0, 255.0, 255.0;
         0.0, 255.0, 255.0, 255.0;
         255.0, 255.0, 255.0, 0.0;
         255.0, 255.0, 0.0 ,0.0];
matrix g= r;
matrix b = r;
save(r,g,b,"load_1.jpg");
```

8.42 save_2.fb

```
matrix r; matrix g; matrix b;
r,g,b = load("save_2.jpg");
int i; int j;
i,j = size(r);
//r[:, 0:j/2] = zeros(i, j/2+1);
//g[:, 0:j/2] = zeros(i, j/2+1);
//b[:, 0:j/2] = zeros(i, j/2+1);
//printf(j/2);
save(r[:i/2, :j/2] ,g[:i/2, 0:j/2] ,b[:i/2, 0:j/2]
   ,"save_2_result.jpg");
//save(r,g,b,"save_2_result.jpg");
```

8.43 scope_1.fb

```
int i = 0;
{
   int j = 5;
   printf(i);printf(j);
   {
      i = 1;
      int j = 6;
      printf(i);printf(j);
      {
         i = 2;
      }
   }
   {
      {
         int i;
         i = 3;
      }
   }
}
printf(i);
//printf(j);//give error variable j not declared.
```

8.44 scope_2.fb

```
int i = 0;
{
   int j = 5;
   printf(i);printf(j);
   {
      i = 1;
      int j = 6;
      printf(i);printf(j);
      {
         i = 2;
         int j = 0;
      }
      printf(j);
   }
   {
      {
          int i;
          i = 3;
      }
      i = 9;
   }
}
printf(i);

//should print: 051669
```

8.45 scope_3.fb

```
int i = 0;
int j;
for (j = 1; j <= 10; j=j+1) {
  i = i + j;
}
printf(i);
printend();
printf(j);
printend();
```

8.46 scope_4.fb

int i = 0; int j; int i = 10; for (j = 1; j ¡= 10; j=j+1) i = i + j; printf(i); printend(); printf(j); printend();

8.47 scope_5.fb

```
int i = 0;
int j;
{
  int i = 10;
  for (j = 1; j <= 10; j=j+1) {
    i = i + j;
  }
  printf(i);
  printend();
  int j = 100;
}

printf(i);
printend();
printf(j);
printend();
```

8.48 semant_assign_1.fb

```
string s = "abc";
printf(s);
//s = 1+1;
s = "a";
printf(s);
```

8.49 semant_assign_2.fb

```
matrix m;
```

```
m = zeros(2,2);
m = 3;
```

8.50 semant_assign_3.fb

```
matrix a = zeros(2,2);
a[1,1]= 2.2;printf(a);
a[0,0] = [2.2];
```

8.51 semant_assign_4.fb

```
matrix a = zeros(3,3);
matrix b = [1.1,2.2;3.3,4.4];
a[1:,:1] = b;
printf(a);
a[:,1:] = b;
```

8.52 semant_assign_5.fb

```
func f() {return 1, 2.2, "str", [1.1;2.2];}
int i; double d; string s; matrix m;
i, s, m = f();
printf(i);printend();
printf(d);printend();
printf(s);printend();
printf(m);printend();
```

8.53 semant_func_2.fb

```
func f() {return;}
f2();
```

8.54 semant_func_3.fb

```
func f(int i, double d, matrix m)
{
   printf(i);printend();
   printf(d);printend();
   printf(m);printend();
}
f(2.2, 2.2, zeros(2,2));
```

8.55 semant_func_rename_1.fb

```
func size(){return;}
1+1;
```

8.56 semant_func_rename_2.fb

```
func f() {return;}
func f() {return 1;}
```

8.57 semant_local_1.fb

```
//matrix a = 12;
int i = "abc";
```

8.58 semant_matrix_1.fb

```
matrix a = zeros(2,2);
matrix b = zeros(3,3);
a.*b;
```

8.59 semant_matrix_2.fb

```
matrix a = zeros(3,3);
printf(a[:, :]);printend();
printf(a[2:, :]);printend();
printf(a[:, 2:]);printend();
printf(a[1:2, 1:2]);printend();
printf(a[:, 1:2]);printend();
printf(a[-1:1,:]);
```

8.60 semant_predicate_1.fb

```
if (2+3) {printf(1);}
```

8.61 semant_predicate_2.fb

```
bool i = true;
while (1) {printf(1);i=false;}
```

8.62 semant_predicate_3.fb

```
int i = 0;
for (;1+2+3;i=i+1)
{
   printf(i);
}
```

8.63 semant_unop_1.fb

```
printf(-3.4);
printf(!4);
```

8.64 plot.fb

```
func factorial (int i)
{
   if (i==1)
   {
      return 1;
   }
   else
   {
      return i * factorial (i-1);
   }
}
func pow(double x, int i)
{
   double ret = 1.0;
   int j;
   for (j = 0; j!=i; j=j+1)
   {
      ret = x * ret;
   }
   return ret;
}
func quad(double a, double b, double c, double x)
{
   return a*x*x+b*x+c;
}
func cubic(double a, double b, double c, double d, double x)
{
   return a*x*x*x+b*x*x+c*x+d;
}
func sin_approx(double a, double x)
{
   double ret = 0.0;
   int i;
   for (i = 0; i != 15; i=i+1)
   {
      ret = ret + pow(x,i*2+1)*pow(-1.0, i)/factorial(i*2+1);
   }
   ret = ret * a;
   return ret;
}
matrix x = zeros(1,201);
```

```
matrix y = zeros(1,201);
int i;
for (i=0; i!= 201; i=i+1)
{
   x[0,i] = -10+i*0.1;
   //y[0,i] = quad(1.0, 0.0, -3.0, x[0,i]);
   //y[0,i] = cubic(0.1, 0.0, -3.0, -5.0, x[0,i]);
   y[0,i] = sin_approx(5.0, x[0,i]);
}
matrix plt_r = 254.0 + zeros(201,201);
matrix plt_g = 254.0 + zeros(201,201);
matrix plt_b = 254.0 + zeros(201,201);
for (i=0; i!= 201; i=i+1)
{
   plt_r[i,101] = 0.0;
   plt_r[101,i] = 0.0;
   plt_g[i,101] = 0.0;
   plt_g[101,i] = 0.0;
   plt_b[i,101] = 0.0;
   plt_b[101,i] = 0.0;
   if (((10-y[0,i])/0.1 <= 200) && ((10-y[0,i])/0.1 >= 0))
   {
      plt_r[double2int((10-y[0,i])/0.1),i] = 0.0;
      plt_g[double2int((10-y[0,i])/0.1),i] = 0.0;
   }
}
save(plt_r, plt_g, plt_b, "plot.jpg");
```

8.65 face_1.fb

```

matrix m;
m = face("d.jpg");
//m = face("b.jpg");
matrix m_r; matrix m_g; matrix m_b;
m_r, m_g, m_b = load("d.jpg");
//m_r, m_g, m_b = load("b.jpg");
double x = m[0,0]; double y = m[1,0]; double l = m[2,0]; double
   w = m[3,0];
int i;
for (i = double2int(x - l/2); i <= double2int(x +l/2); i = i+1)
```

```
{
    m_g[i, double2int(y-w/2-2):double2int(y-w/2+2)] = (255.0-zeros(1,5));
    m_b[i, double2int(y-w/2-2):double2int(y-w/2+2)] = (255.0-zeros(1,5));
    m_r[i, double2int(y-w/2-2):double2int(y-w/2+2)] = zeros(1,5);
    m_g[i, double2int(y+w/2-2):double2int(y+w/2+2)] = (255.0-zeros(1,5));
    m_b[i, double2int(y+w/2-2):double2int(y+w/2+2)] = (255.0-zeros(1,5));
    m_r[i, double2int(y+w/2-2):double2int(y+w/2+2)] = zeros(1,5);
}
for (i = double2int(y - w/2); i <= double2int(y +w/2); i = i+1)
{
    m_g[double2int(x-l/2-2):double2int(x-l/2+2), i] = (255.0-zeros(5,1));
    m_b[double2int(x-l/2-2):double2int(x-l/2+2), i] = (255.0-zeros(5,1));
    m_r[double2int(x-l/2-2):double2int(x-l/2+2), i] = zeros(5,1);
    m_g[double2int(x+l/2-2):double2int(x+l/2+2), i] = (255.0-zeros(5,1));
    m_b[double2int(x+l/2-2):double2int(x+l/2+2), i] = (255.0-zeros(5,1));
    m_r[double2int(x+l/2-2):double2int(x+l/2+2), i] = zeros(5,1);
}
//save(m_r, m_g, m_b, "face_1_result.jpg");
save(m_r, m_g, m_b, "face_2_result.jpg");
```

8.66 sharpen.fb

```
matrix t_r; matrix t_g; matrix t_b;
t_r,t_g,t_b = load("sbird2.jpg");
matrix r_r; matrix r_g; matrix r_b;
//printf(t_r);
//printf(t_g);
matrix s = [0.0, -1.0, 0.0;
        -1.0, 5.0, -1.0;
         0.0, -1.0, 0.0];
//int i;int j;
//i,j = size(t_r);
//printf(i);printf(j);
```

```
r_r = t_r $ s;
r_g = t_g $ s;
r_b = t_b $ s;
save(r_r, r_g, r_b, "sbird_result.jpg");
```