MiniMat: Matrix Language in OCaml LLVM

Terence Lim - `tl2735@columbia.edu`

August 28, 2016

Contents

Listings

1 Introduction

MiniMat is a typed, imperative language to support matrix-based programming, which contains a core set of primitives that can be assembled into more complicated abstractions for matrix expressions, linear algebraic formulas and statistical algorithms. MiniMat is aimed at programmers from technical domains who work primarily with matrix expressions. It treats large data types such as a two-dimensional matrix as a primitive data type and accepts syntax so that matrices are easy to initialize, subset, combine and reshape with *bracket* expressions. Such expressions as well as all matrix operators are implemented by helper functions written in the MiniMat language itself.

The compiler front end components (lexer, parser, semantic analysis and intermediate code generation) are written in OCaml, OCamllex, OCamlyacc and OCaml Llvm, and translates source language into LLVM IR target code.

The next section provides a short tutorial. The language reference manual is in section 3. Section 4 presents the project plan, while section 5 describes the architectural design. Section 6 contains the test suite and representative source language programs. Section 7 reflects on lessons learned, while the Appendix attaches a complete code listing of the translator (written in OCaml), as well as support libraries (coded in MiniMat language) that help implement matrix expressions, operators and functions.

1.1 Goals

The language specification aims to achieve the following goals:

1.1.1 Flexible matrix notation

It enables matrix bracket expressions, so that it is easy to write matrix expressions resembling mathematical matrix notation to construct and augment matrix data objects, simultaneously retrieve or place values in multiple locations within a matrix, and apply matrix operators.

1.1.2 Uncluttered

It focused on providing a small set of primitives that is absolutely necessary, rather than a heavy set that is poorly implemented.

1.1.3 Potential

More powerful abstractions can then be coded entirely in the MiniMat language itself. The first layer of functions, coded in MiniMat, help implement matrix bracket and reference expressions. All matrix operators are then bound to and implemented with a next layer of helper functions. These routines all check that the matrix sizes and index positions of their operands are mutually consistent. From these, linear algebraic formulas can be expressed with familiar mathematical matrix notation: an extensive library of matrix mathematical functions is provided. More sophisticated statistical algorithms such as logistic regression can then be coded up.

2 Language Tutorial

2.1 Installation

To install, unpack the package in a new directory, and run `make` from the base directory. This builds the `minimat` compiler in the `./src` subdirectory; runs the test suites in the `./tests` subdirectory; and compiles the example programs and *gnuplot*[1] external library. The directory contents are:.

`./src/`	OCaml source programs to build `minimat` compiler
`./include/`	support library source files coded in MiniMat language
`./tests/`	test suite of MiniMat source programs and expected output
`./examples/`	MiniMat tutorial and sample source programs
`./include/gnuplot_i/`	external gnuplot C API library for visualizing results
`./doc/`	this manual and other documentation

2.2 Compiling a MiniMat source program

Since the MiniMat compiler generates LLVM IR target code, a convenient script `mmc` is provided[2] to wrap the translation (of MiniMat to LLVM IR), compilation[3] and linking[4] tasks. For example, create a simple MiniMat source program such as *hello.mm* below, then compile it with `mmc` and run the resultant executable *hello*. Note that MiniMat programs require and begin execution with the `int`-typed function named `main()`.

```
$ ./mmc hello.m
$ ./hello
```

Listing 1: "Hello, world" in MiniMat

```
int main() {
  printstring("hello, world!\n");
}
```

2.3 Tutorial

MiniMat is an imperative, typed language that is a superset of MicroC introduced in class, which in turn is a subset of C. This tutorial describes MiniMat statements for defining

[1]To use the gnuplot external C routines, you must also have gnuplot installed and working on your machine, accessible from the user account you are using. If gnuplot is not installed, run `make noplot` option instead from the base directory to install the package.

[2]If gnuplot is not installed, then use the script `mmc-noplot` instead to compile your MiniMat source file.

[3]Requires `llc`.

[4]Requires `gcc`.

matrix literals and functions, and using matrix operators and expressions. The output from a program combining the statement examples is presented at the end of the section.

2.3.1 Defining a matrix literal

A `matrix` type literal in MiniMat comprises "rectangular" rows of columns of floating point values. It is defined with semi-colon-separated rows of comma-separated floating point numbers, all enclosed by a set of square brackets. Hence the following MiniMat statements declare and define a 2-row by 4-column matrix comprised of the values 1.0 through 8.0. Note that the ending semi-colon (to terminate the last row) before the closing bracket is required. The function `printmat` simply pretty-prints its single matrix argument – this function is provided in the support routines files (all of which are coded in the MiniMat language itself) in the `./include/` subdirectory.

```
  matrix a;
  a = [ 1.0, 2.0, 3.0, 4.0;
        5.0, 6.0, 7.0, 8.0; ];
  printmat(a);
```

2.3.2 Matrix size attributes

The `rows` and `cols` functions return the dimensional attributes of a single matrix argument as an integer value, which can be printed with the `printint` function. The dimensions of a matrix are stored internally with the matrix object, and do not need to be tracked explicitly when calling matrix functions or operators.

```
  printint(rows(a));
  printint(cols(a));
  println();             /* print a newline */
```

2.3.3 Defining a matrix function

Functions in MiniMat are defined separately outside the `main()` function. The following defines a function named *twos* which takes two `int` arguments and returns a new `matrix` whose elements all have floating point value of 2.0 (note that floating point literals must always include a decimal point). It uses the modifier `new` to construct a new `matrix` of the number of rows and columns given in its arguments, with values initialized to 0.0 by default. It then adds 2.0 to these values and returns the new matrix.

```
matrix twos(int r,int c) { /* return r row by c column matrix of 2's */
  return new matrix(r, c) + [2.0;];
}
```

2.3.4 Matrix function arguments

Matrices can be supplied as arguments to functions. Specifically, the pointer to contents of a matrix is passed by value. That means the address of the matrix contents pointed to is copied and passed to the function. Hence if the value of the matrix pointer is changed in the called function body, that replacement will not be reflected in the caller's matrix address which will still point to the old object: i.e. the original matrix in the caller will not be replaced by an assignment statement in the callee. But values of any of the cell contents of the matrix can be modified. For example, in the body of following function `changeit`, its first argument is reassigned to the empty matrix (note that the semi-colon before the closing bracket, to terminate the last (possibly empty) matrix row, is required) and its second argument is modified: the value in its first row and second column is changed to 42.0. A matrix column or row index position begins counting at 0, and is referenced with comma-separated row and column integers enclosed by square brackets (as described later, multiple positions in a matrix can be simultaneously referenced by using sequences as indices). Also, the transpose of a matrix `A` can be obtained either with the `A'` postfix unary operator, or by calling its associated helper function `mtransp(A)` instead.

```
matrix changeit(matrix replaceme, matrix modifyme) {
  replaceme = [;];           /* attempt to replace with empty matrix */
  modifyme[0,1] = [42.0;];   /* change value in row 0, column 1  */
  return modifyme';          /* matrix transpose postfix operator */
}
```

When the function `changeit` defined above is called, the first argument passed, the matrix `a`, will not be replaced in the caller, but the contents of the second argument, matrix `b`, is modified.

```
  matrix b;
  matrix c;
  c = changeit(a, b);  /* matrix args are pointers passed by value */
  printmat(a);         /* original matrix will not be replaced */
  printmat(b);         /* but contents may be modified */
  printmat(c);
```

2.3.5 Matrix operators

Addition, multiplication and other operators can be applied to matrices. For example, the following statement first multiplies each element of the matrix `a` by itself, then adds each element of the resultant matrix to `a` again. Multiplicative binary operators have higher precedence than additive.

```
  b = a + a .* a;
  printmat(b);
```

2.3.6 Augmenting matrices

The bracket expressions to construct matrix literals can also nest smaller matrices, not just floating point literals or variables, but require the sizes of adjacent components to be compatible. The comma operator has higher precedence than semi-colon, hence all adjacent columns are combined first, and then the resultant rows are concatenated. Adjacent comma-separated components must have the same number of rows, since the comma separator essentially concatenates horizontally. Subsequently, semi-colon-separated components must have the same number of columns, since the semi-colon separator essentially concatenates the components vertically. The matrix construction statement below is presented with operands spaced out to approximate their respective positions in the matrix. In the second and third lines, `a` (a 2-row × 4-column matrix) must have the same number of rows as its adjacent matrix literal expression (which has 2 rows and 3 columns), as do `new matrix(2, 3)` and matrix `b` (which has 2 rows and 4 columns) in the fourth line. All three sets of lines, separated by semi-colons, have the same number of columns (7).

```
c = [-1.0, -2.0, -3.0, -4.0,-5.0, -6.0, -7.0;
     a,                         [11.0, 12.0, 13.0;
                                 14.0, 15.0, 16.0; ];
     new matrix(2, 3),  b;];
printmat(c);
```

2.3.7 Defining a sequence literal

The `sequence` data type is just a list of integers. It is most useful in MiniMat for helping select a subset of a matrix with sequences of position values. A `sequence` literal is defined as a comma-separated list of `int` values or other shorter `sequence`s, enclosed by square brackets. The colon operator can also help construct a sequence given the first, step, and end values as operands. When the step size has default value of 1, the double-colon operator can be used, without explicitly specifying the step size value.

```
sequence x;
sequence y;
sequence z;
x = 1:2:5;      /* colon stride operator. constructs [1, 3, 5] */
y = -5::5;      /* double-colon operator: implicit stride length 1 */
z = [1, 2, x]; /* comma-separated ints or sequences, in brackets */
printseq(x);
printseq(y);
printseq(z);
```

2.3.8 Matrix references

The contents of a matrix cell is referenced by its row and column, separated by a comma, in square brackets. When a `sequence` instead of just an `int` is used to reference the matrix, all the corresponding matrix rows and columns listed in the sequence(s) are selected. This reference method can also be used for assigning values to multiple locations within a matrix: for example, in the second assignment statement below, all the rows and columns of the target matrix `c` that are specified by its respective sequence arguments (specifically, the first sequence expression selects all its rows, while the second selects just those columns listed in `x`) are replaced by the corresponding floating values from source matrix `d`; all other values of `c` not in row and column positions contained in the respective sequence indices are unchanged.

```
matrix d;
```

```
d = c[0::rows(c) - 1, x];    /* select multiple positions from c */
printmat(d);

d = d * [10.0;];
c[0::rows(c) - 1, x] = d;    /* assign to multiple positions in c */
printmat(c);
```

2.3.9 Loops and conditionals

The following statements provide an example of using a for-loop, with an integer counter, to construct a sequence of Fibonacci numbers. The keyword `new` constructs a `sequence` object with argument length comprising integer values initialized to 0 by default. An element of a `sequence` can be referenced with its integer position (or sequence of positions) enclosed in brackets. The `end` function is provided in the support library to return the index of the last position of its `sequence` argument, while `printseq` pretty-prints the resultant `sequence`.

```
int i;
sequence fib;
fib = new sequence(10);
fib[0] = [0];
fib[1] = [1];
for(i = 2; i < length(fib); i = i + 1) {
  fib[i] = fib[i - 1] + fib[i - 2];
}
printseq(fib[end(fib)]);
```

2.3.10 Additional data types

MiniMat also implements the data types `string` (which is a null-terminated list of characters), `float` and `bool`, with associated operators and functions. The input functions `next()`, `nextint()` and `nextfloat()` reads the next value of the respective types from standard input. Similarly, the functions `printint`, `printfloat`, `printbool` and `printstring` print their argument of the respective types to standard output. Values of the same type can be compared with relational operators (though conversions between types is not automatic but require explicit typecast functions – see language reference manual): For example, the string equality infix operator `==` returns a boolean value `true` if both string operands have identical character values.

```
printstring("Type in a float: ");
f = nextfloat();
printstring("Float value read in = ");
printfloat(f);

printstring("\nType in an int: ");
i = nextint();
printstring("Int value read in = ");
printint(i);
```

```
 printstring("\nType in the string (without quotes) \"hello\" : ");
 s = next();
 B = (s == "hello");
 printbool(B);
```

Listing 2: Complete program listing for MiniMat tutorial

```
matrix twos(int r, int c) {   /* return r row by c column matrix of 2's */
  return new matrix(r, c) + [2.0;];
}

matrix changeit(matrix replaceme, matrix modifyme) {
  replaceme = [;];          /* attempt to replace with empty matrix */
  modifyme[0,1] = [42.0;]; /* change value in row 0, column 1 of matrix */
  return modifyme';         /* matrix transpose postfix operator */
}

int main() {
  matrix a;
  matrix b;
  matrix c;
  matrix d;
  sequence x;
  sequence y;
  sequence z;
  sequence fib;
  int i;
  float f;
  string s;
  bool B;

  printstring("Define a matrix literal:\n");
  a = [ 1.0, 2.0, 3.0, 4.0;
        5.0, 6.0, 7.0, 8.0; ];
  printmat(a);

  printstring("Matrix row and column attributes:\n");
  printint(rows(a));
  printint(cols(a));
  println();          /* print newline */

  printstring("Define a matrix function to return a matrix of 2's\n");
  b = twos(4, 2);
  printmat(b);

  printstring("Matrix function arguments\n");
  c = changeit(a, b);  /* matrix args are pointers passed by value */
  printmat(a);         /* original matrix will not be replaced */
  printmat(b);         /* but contents may be modified */
  printmat(c);

  printstring("Matrix operators -- addition, dot-multiply:\n");
  b = a + a .* a;
  printmat(b);

  printstring("Horizontally and vertically augment matrix:\n");
  c = [-1.0, -2.0, -3.0, -4.0,-5.0, -6.0, -7.0;
       a,                    [11.0, 12.0, 13.0;
                              14.0, 15.0, 16.0; ];
       new matrix(2, 3),  b;];
  printmat(c);

  printstring("Construct sequence literals:\n");
  x = 1:2:5;      /* colon stride operator, constructss [1, 3, 5] */
  y = -5::5;      /* double-colon operator: implicit stride length 1 */
  z = [1, 2, x]; /* comma-separated ints or sequences, in brackets */
  printseq(x);
  printseq(y);
  printseq(z);

  printstring("Subselect alternate columns of matrix with sequence index:\n");
  d = c[0::rows(c) - 1, x];     /* select multiple positions from c */
  printmat(d);

  printstring("Assign values into subset of a matrix with sequence index:\n");
  d = d * [10.0;];
  c[0::rows(c) - 1, x] = d;     /* assign to multiple positions in c */
  printmat(c);

  printstring("Loop to construct a sequence of Fibonacci numbers:\n");
  fib = new sequence(10);
  fib[0] = [0];
  fib[1] = [1];
  for(i = 2; i < length(fib); i = i + 1) {
    fib[i] = fib[i - 1] + fib[i - 2];
  }
  printseq(fib[end(fib)]);

  printstring("Type in a float: ");
  f = nextfloat();
  printstring("Float value read in = ");
  printfloat(f);

  printstring("\nType in an int: ");
  i = nextint();
  printstring("Int value read in = ");
  printint(i);

  printstring("\nType in the string (without quotes) \"hello\" : ");
  s = next();
  B = (s == "hello");
  printbool(B);
}
```

Output log of tutorial program

```
Define a matrix literal:
 [2 x 4 float]
  1.00 2.00 3.00 4.00
  5.00 6.00 7.00 8.00
Matrix row and column attributes:
 2 4
Define a matrix function to return a matrix of 2's
 [4 x 2 float]
  2.00 2.00
  2.00 2.00
  2.00 2.00
  2.00 2.00
Matrix function arguments
 [2 x 4 float]
  1.00 2.00 3.00 4.00
  5.00 6.00 7.00 8.00
[4 x 2 float]
  2.00 42.00
  2.00 2.00
  2.00 2.00
  2.00 2.00
[2 x 4 float]
  2.00 2.00 2.00 2.00
 42.00 2.00 2.00 2.00
Matrix operators -- addition, dot-multiply:
 [2 x 4 float]
  2.00 6.00 12.00 20.00
 30.00 42.00 56.00 72.00
Horizontally and vertically augment matrix:
 [5 x 7 float]
 -1.00 -2.00 -3.00 -4.00 -5.00 -6.00 -7.00
  1.00 2.00 3.00 4.00 11.00 12.00 13.00
  5.00 6.00 7.00 8.00 14.00 15.00 16.00
  0.00 0.00 0.00 2.00 6.00 12.00 20.00
  0.00 0.00 0.00 30.00 42.00 56.00 72.00
Construct sequence literals:
 [3 int]
1 3 5
[11 int]
-5 -4 -3 -2 -1 0 1 2 3 4 5
[5 int]
1 2 1 3 5
Subselect alternate columns of matrix with sequence index:
 [5 x 3 float]
 -2.00 -4.00 -6.00
  2.00 4.00 12.00
  6.00 8.00 15.00
  0.00 2.00 12.00
  0.00 30.00 56.00
Assign values into subset of a matrix with sequence index:
 [5 x 7 float]
 -1.00 -20.00 -3.00 -40.00 -5.00 -60.00 -7.00
  1.00 20.00 3.00 40.00 11.00 120.00 13.00
  5.00 60.00 7.00 80.00 14.00 150.00 16.00
  0.00 0.00 0.00 20.00 6.00 120.00 20.00
  0.00 0.00 0.00 300.00 42.00 560.00 72.00
Loop to construct a sequence of Fibonacci numbers:
 [1 int]
34
Type in a float: Float value read in = 3.14
Type in an int: Int value read in = 42
Type in the string (without quotes) "hello" : 1
```

3 Language Reference Manual

This section describes the MiniMat language specification.

3.1 Lexical Conventions

A program is reduced to a stream of tokens by the initial lexical analysis or scanning phase.

3.1.1 Tokens

There are five classes of tokens: identifiers, keywords, literals, operators, and separators. Blanks, tabs, newlines, form feeds and comments (as described below) are ignored except to separate tokens.

3.1.2 Comments

The characters `/*` and `*/` demarcate a comment. Comments do not nest, and can be split across multiple lines.

3.1.3 Identifiers

An identifier is a sequence of letters, digits and the underscore _ character. The first character must be a letter.

3.1.4 Keywords

The following identifiers are reserved for use as keywords and implemented as built-in instructions:

`bool`	`int`	`float`	`string`	`void`	`handle`
`matrix`	`sequence`	`external`	`constant`	`true`	`false`
`if`	`else`	`for`	`while`	`return`	`new`
`length`	`cols`	`float_of_int`	`int_of_float`	`int_of_seq`	`float_of_mat`

Additionally, matrix bracket expressions for construction, assignment and selection as well as matrix operators are bound to and implemented with helper functions written in the MiniMat language and included in standard library files. The names and descriptions of these library functions, all of which can be called standalone, are listed at the end of this section.

3.1.5 Literals

There are four types of literals.

```
literal:
  integer-literal
  floating-literal
  boolean-literal
  str-literal
```

- **Integer**: An integer literal, with data type `int`, consists of a sequence of digits.
- **Floating**: A floating literal, with data type `float`, consists of an optional integer part, a decimal point and a fraction part. The integer and fraction parts both consist of a sequence of digits. The integer part may be missing, and both the decimal point and fraction part are required.
- **Boolean**: A boolean literal is one of two values `true` or `false`, of data type `bool`.
- **String**: A string literal is a sequence of characters surrounded by double quotes, of data type `string`. A null byte is appended to the string C-style so that functions for printing or comparing strings can scan to find its end.

3.2 Data types

There are three *small*, three *large*, and two other data types.

```
type-name: one of
  int float bool matrix sequence string handle void
```

3.2.1 Small types

- `int`: A signed integer has 32-bit size.
- `float`: A floating point value has 64-bit size (i.e. a *double* in C-parlance).
- `bool`: A boolean requires 1-bit size.

3.2.2 Large types

- `matrix`: A two-dimensional matrix of floating point values, laid out in row-major order – i.e. sequential values on the same row are stored adjacent to each other. Size information, such as the number of rows and columns, is stored with the object.
- `sequence`: A list of integer values. Size information such as length is stored with the object. This data type is most helpful for providing a list of index positions to access subsets of a `matrix` or another `sequence`.
- `string`: Strings are stored as null-terminated sequences of characters, up to 256 characters length (including terminal null).

The `new` modifier constructs a new object of the type of its operand, which may be a `matrix`, `sequence` or `string`, and allocates storage space for it. A `new matrix`*(int, int)* requires two arguments specifying the number of rows and number of columns as integers: its contents are initialized to zero by default. Similarly, `new sequence`(*int*) and `new string`() require one argument specifying the sequence length as an integer and no arguments respectively. Alternatively, a `new string`(*string*,...) can be initialized by providing a format string as its first argument, followed by a variable list of arguments to be used by conversion specifiers in the format string. The specification of the format string and the argument list follows exactly that implemented by the *printf* group of system functions – see system `man` pages for specification details.

```
type-constructor:
  new type-name ( optional-argument-expression-list )
```

3.2.3 Other types

- `handle`: This data type is only utilized when calling external C library functions which may need to pass around a pointer to an external object. It is a 64-bit value representing a memory address.
- `void`: Specifying an empty set of values, it is used as the type returned by functions that generate no value.

3.2.4 Conversions

Generally, operators will not cause conversion of the value of an operand from one type to another, i.e. there is no automatic type casting. MiniMat provides four primitive built-in conversion functions `float_of_int`, `int_of_float`, `int_of_seq` and `float_of_mat`. When the value of a floating type is converted to integral type, the fractional part is discarded.

3.3 Expressions

The precedence of expression operators is the same as the order of the following major subsections, highest precedence first. Within each subsection, the operators have the same precedence, with left- or right-associativity as specified.

3.3.1 Primary Expressions

Primary expressions are identifiers, literals, type constructors or expressions in parentheses.

```
primary-expression:
  identifier
  literal
  mat-literal
  seq-literal
  type-constructor
  ( expression )
```

An identifier is a primary expression provided it has been suitably declared as a constant; local or external function; or global, local or function argument variable. Its type is specified by its declaration.

A parenthesized expression is a primary expression whose type and value are identical to the unadorned expression.

3.3.2 Matrix Literal Expressions

```
mat-literal:
  [ optional-mat-row-list ; ]

mat-row-list:
  mat-expression-list
  mat-row-list ; mat-expression-list

expression-list:
  expression
  expression-list , expression
```

A matrix literal, of data type `matrix`, is defined as semi-colon-terminated rows of comma-separated floating points or other matrices (or their identifiers), surrounded by square brackets.

This bracket expression syntax can also be used to augment a matrix by nesting other (smaller) matrices, not just floating point literals. The sizes of adjacent components must be compatible. The comma operator has higher precedence than semi-colon, hence all adjacent columns are combined first, and then the resultant rows are combined. Adjacent comma-separated components must have the same number of rows, since the comma separator essentially concatenates rows horizontally. Subsequently, semi-colon-separated components must have the same number of columns, since the semi-colon separator essentially concatenates the columns vertically.

3.3.3 Sequence Literal Expressions

```
seq-literal:
  [ optional-seq-expression-list ]
```

A `sequence` literal can be defined as a comma-separated list of integers or shorter sequences (or their identifiers), surrounded by square brackets.

A matrix literal, even if empty, always ends with a semi-colon just before the closing bracket; a sequence never does. A matrix requires floating point numbers, which always contain a decimal point; a sequence requires integers, which do not.

3.3.4 Postfix Expressions

```
postfix-expression:
  primary-expression
  mat-identifier [ row-index-expression , column-index-expression ]
  seq-identifier [ index-expression ]
  function-identifier ( optional-argument-expression-list )
  postfix-expression '
```

- **Matrix References**

  An identifier followed by two comma-separated expressions in square brackets is a postfix expression denoting a subscripted `matrix` reference. The two expressions together comprise the row-column index method to reference positions in a `matrix`. They can each either be of `int` type (identifying a single column or row) or `sequence` type (representing several columns or rows of the matrix simultaneously). This reference method can be used to either assign or select values in multiple positions of a matrix, i.e. it can generate both modifiable l-values suitable for assignment to, as well as r-values.

- **Sequence References**

  An identifier followed by a single expression in square brackets is a postfix expression denoting a subscripted reference to positions in a `sequence`. The bracketed expression can either be of `int` type (representing a single element) or `sequence` type (identifying a subset of elements). This reference method can be used to either assign or place values in multiple positions of a sequence.

- **Function Calls**

  A function call is a functional designator identifier, followed by parentheses containing a possibly empty, comma-separated list of arguments to the function. Arguments are completely evaluated before the function is entered. Recursive calls are permitted.

  In preparing for the call to a function, a copy is made of *small* data type arguments: all argument-passing of `int, float, bool` and `handle` is by value. With large data objects `matrix, sequence` and `string`, a copy of the pointer to contents of the object is passed as the argument. Hence if the value of the object pointer in the called function body is changed, that replacement will not be reflected in the caller's address value which will still point to the old object: i.e. the original matrix in the caller will not be replaced by an assignment statement in the callee. But values of the contents in the object can be modified.

- **Transpose** (`'`): This is a postfix expresson that reshapes its `matrix` argument to transposed form, swapping its rows and columns. The operand, and hence result, are of type `matrix`. This operator is left-associative.

3.3.5 Unary Operators

Unary operators are right-associative.

```
unary-expression:
  postfix-expression
  ! unary-expression
  - unary-expression
  length ( expression )
  cols ( expression )
  cast-name ( cast-expression )

cast-name: one of
  int_of_float float_of_int int_of_seq float_of_mat
```

- **Unary Minus** (`-`): The operand must be `int, float, sequence` or `matrix`, and the result is the negative of (all elements of) its operand.
- **Logical Negation** (`!`): The operand must have boolean type, and the result is the opposite of (i.e. *not*) the operand.
- **Sizeof Operators**

  There are two built-in sizeof operators. The `length` operator yields the actual number of items in its operand, which can be of type `matrix` or `sequence`. The `cols` operator yields the actual number of columns of its `matrix` operand (note that since rows can be computed implicitly from length and columns, it is not provided as a built-in but can be coded as a support function).
- **Cast Operators** These built-in operators converts the value of its operand to and from the respective types.

3.3.6 Power Operator

```
power-expression:
  unary-expression
  power-expression ^ unary-expression
  power-expression .^ unary-expression
```

- **Power** (`^`): This operator is left-associative. When the left operand has type `float`, the right operand must have type `float` which is the power by which the left operand is raised to. The right operand `k` must have type `int` when the left operand has type `matrix`, in which case it is mat-multiplied by itself `k` times.
- **Element-by-element Power** (`.^`): The left and right operands must have type `matrix`. This operator is left-associative, and yields the value of each element of the first operand raised to the specified power of the corresponding element in the second operand. If either operand only has one element, it is first replicated to form a matrix of the same size as the other operand.

3.3.7 Multiplicative Operators

```
multiplicative-expression:
  power-expression
  multiplicative-expression * power-expression
  multiplicative-expression / power-expression
  multiplicative-expression % power-expression
  multiplicative-expression .* power-expression
  multiplicative-expression ./ power-expression
  multiplicative-expression .% power-expression
```

- **Multiplicative** (`* / %`): The multiplication, division and remainder operators are left- associative.

  The two operands must have the same type. When they are of type `float` or `int`, the binary `*` operator yields the product, while the binary `/` yields the quotient, and the `%`

operator the remainder of the division of the first operand by the second; if the second operand is 0, the result is undefined.

When the operands are both of type `sequence`, the binary operators are applied pairwise by element, and the result is returned in a `sequence` of the same length.

When the operands are both of type `matrix`, the binary `*` operator denotes matrix multiplication. The binary `/` operator yields the coefficients of a least squares regression of the left operand on the right. The binary `%` operator yields the deviations from this regression.

- **Element-by-element Multiplicative** (`.* ./ .%`): The element-by-element multiplication, division and remainder operators are left-associative. The left and right operands must have type `matrix`. The binary `.*` operator yields the product, while the binary `./` yields the quotient, and the `.%` operator the remainder of the division of each element of the first operand with the second operand. If either operand only has one element, it is first replicated to form a matrix of the same size as the other operand.

3.3.8 Additive Operators

```
additive-expression:
  multiplicative-expression
  additive-expression + multiplicative-expression
  additive-expression - multiplicative-expression
```

- **Additive operators** (`+ -`): The addition and subtraction operators are left-associative.

  The two operands must have the same type. When they are of type `float` or `int`, the binary `+` operator yields the sum, while the binary `-` yields the difference of the two operands.

  When the operands are of type `sequence` or `matrix`, the binary operators are applied pairwise by element, and the result is returned in a `sequence` or `matrix` of the same size. If either operand only has one element, it is first replicated to form a matrix or sequence of the same size as the other operand.

3.3.9 Colon Operators

```
colon-expression:
  additive-expression
  additive-expression : additive-expression : additive-expression
  additive-expression :: additive-expression
```

- **Colon Stride Operator** (`:    :`): This operator takes three `int` operands (with a colon separating each pair of adjacent operands) and yields a `sequence` listing integer values ranging between the first to third operands, with the second operand representing the stride to skip over. The stride can be negative, in which case the first operand must be at least as large as the third operand. A (default) stride value of 1 can be left out of the expression, using a double-colon operator, i.e. two colons together which separate the beginning and ending range values.

3.3.10 Relational Operators

Relational operators are left-associative.

```
relational-expression:
  colon-expression:
  relational-expression > colon-expression
  relational-expression >= colon-expression
  relational-expression < colon-expression
  relational-expression <= colon-expression
```

- **Ordering Operators** (`< > <= >=`): The less than, greater than, less or equal, and greater or equal binary operators all yield the boolean value of the specified relation, when the operands are of type `int` or `float`.

  When the operands are of type `matrix`, the binary operators are applied pairwise element-by-element; the linear indices of pairwise elements where the specified relation is true are returned in a `sequence`. If either operand only has one element, it is first replicated to form a matrix of the same size as the other operand.

  When the operands are of type `string`, the strings' character contents are compared, and yields the boolean value of the specified relation in dictionary order.

3.3.11 Equality Operators

Equality operators are left-associative.

```
equality-expression:
  relational-expression:
  equality-expression == relational-expression
  equality-expression != relational-expression
```

- **Equality Operators** (`== !=`): The equal and not equal binary operators all yield the boolean value of the specified relation, when the operands are of type `int`, `float` or `bool`.

  When the operands are of type `matrix`, the binary operators are applied pairwise element-by-element; the linear indices of pairwise elements where the specified relation is true are returned in a `sequence`. If either operand only has one element, it is first replicated to form a matrix of the same size as the other operand.

  When the operands are of type `string`, the strings' character contents are compared: strings are not equal when their respective characters in any one position index are not the same.

3.3.12 Logical AND Operator

```
logical-AND-expression:
  equality-expression
  logical-AND-expression && equality-relation
```

- (`&&`): This left-associative operator returns true if both its boolean operands are also true. Both operands are evaluated, regardless of the value of the first.

3.3.13 Logical OR Operator

```
logical-OR-expression:
  logical-AND-expression
  logical-OR-expression || logical-AND-expression
```

- (`||`): This left-associative operator returns true if either of its boolean operands are also true. Both operands are evaluated, regardless of the value of the first.

3.3.14 Assignment

```
assignment-expression:
  logical-OR-expression
  unary-expression = assignment-expression
```

- **Assignment operator** (`=`): This operator is right-associative. The left operand must be a properly declared identifier or a selection from a matrix or sequence reference. The value of the right expression replaces that of the object, or selected positions in the matrix or sequence, referred to by the left operand. When the right operand is an identifier with type `matrix`, `sequence` or `string`, a new copy of its values is made and assigned to the left operand.

3.4 Declarations

Declarations specify the interpretation given to each identifier. Declarations that also reserve storage are called definitions.

```
declaration:
  type-name variable-identifier
```

Declarations may only appear either at the beginning of the body of a function definition ("local variables") or outside of any function definition ("global variables").

3.4.1 Type Specifiers

The type-names are `int float bool sequence matrix string handle void`. One type-name and one identifier name are given in each declaration.

3.5 Statements

Statements can be of several types, and are executed in sequence.

```
statement:
  expression-statement ;
  compound-statement
  selection-statement
  iteration-statement
  retn-statement
```

3.5.1 Expression Statements

Most statements are expression statements, such as assignments or function calls.

3.5.2 Compound Statements

The compound statement, or block, can be used to execute several statements where one is expected. The bodies of a function definition or `while/for` loops are a compound statement, as is any list of statements enclosed within braces { }. Only function definition bodies can include an optional declaration list at the beginning.

```
compound-statement:
  { optional-local-declaration-list optional-statement-list }

local-declaration-list:
  declaration ;
  local-declaration-list declaration ;

statement-list:
  statement
  statement-list statement
```

3.5.3 Selection Statements

The `if` statement chooses a flow of control. It has two forms: `if (`*expression*`)` *statement* or `if (`*expression*`)` *statement* `else` *statement.* If the expression, which must have `bool` type when evalated, is `true` then the first substatement ie executed. In the second form, the second substatement is executed if the expression is `false`. The `else` ambiguity is resolved by connecting to the last encountered `else`-less `if`.

```
selection-statement:
  if ( expression ) statement
  if ( expression ) statement else statement
```

3.5.4 Iteration Statements

The two forms of iteration statements specify looping.

```
iteration-statement:
  while { expression } statement
  for ( optional-expression ; expression ; optional-expression ) statement
```

The `while` statement has the form `while (`*expression*`)` *statement.* The substatement is executed repeatedly so long as the value of the expression remains `true`; the expression must evaluate to a `bool` type.

The `for` statement has the form `for (`*expression1 ; expression2; expression3*`)` *statement.* The first expression is evaluated once to initialize for the loop, and can have any type. The second expression must have `bool` type; it is evaluated before each iteration, and if it becomes `false`, the loop is terminated. The third expression is evaluated after each iteration, and thus specifies a re-initialization for the loop; there is no restriction on its type. The first and third expressions may be dropped, but the second expression is required.

3.5.5 Return Statement

```
retn-statement:
  return optional-expression
```

A function returns to its caller by the `return` statement. When followed by an expression, the value is returned to the caller of the function. A `return` statement, if any, must be the last statement in a function body. Flowing off the end of a function is equivalent to a return with no expression; in either case, the returned value is undefined.

3.6 Program Unit

A program provided to the MiniMat compiler consists of a sequence of declaration units which are either global declarations, function definitions, external function declarations, or global constant definitions.

```
declaration-unit:
  function-definition
  extern-function-declaration
  global-declaration
  global-const-definition
```

3.6.1 Function Definitions

Function definitions have the form

```
function-definition:
  type-name function-identifier ( optional-formal-declaration-arguments )
      compound-statement

formal-declaration-arguments:
  declaration
  formal-declaration-arguments , declaration
```

A function may return any data type, but not a function. By convention, the program entry point is an `int` function named `main()`.

3.6.2 External Function Declarations

External C functions may be used within MiniMat programs, assuming they are loaded in the linker stage, after declaring their function prototypes with the modifier `external`. The following lists the equivalence of MiniMat data types to external (C-language) data types: int (int32 t), float (double), string (char *), sequence (int32 t *), matrix (double *), handle (void *).

```
extern-function-declaration:
  external type-name extern-identifier ( optional-formal-declaration-arguments );
```

3.6.3 Global Variables Declaration

Values of globally-declared variables can be accessed and changed by any function.

```
global-declaration:
  declaration ;
```

3.6.4 Global Constants

Global constants, defined with the modifier *constant*, are globally-defined variables that are each assigned a value at compile-time which never change during execution. The value can be specified by any expression that comprises literals and operators, but not other identifiers. Constant values can be accessed by any function, but cannot be changed. However, their declarations are suspended when global or local variables in scope are declared with the same name.

```
global-const-definition:
  constant type-name const-identifier = expression ;
```

3.7 Lexical Scope

Identifiers may specify either functions or parameters (i.e. functional arguments, local variables, global variables, and constants). The same identifier name may be used for each of these two purposes and will not interfere with one another.

The scope of a parameter of a function definition begins at the start of the block defining the function, persists through the function and ends at the end of the declarator. If an identifier is declared at the head of a function (i.e. a local variable), any declaration of the identifier outside the function as a global variable or constant is suspended until the end of the function. If a global variable is explicitly declared, then any definition of a global constant with the same identifier name is suspended.

3.8 Grammar

The full grammar is collected below:

```
declaration-unit:
  function-definition
  extern-function-declaration
  global-declaration
  global-const-definition

function-definition:
  type-name function-identifier ( optional-formal-declaration-arguments )
      compound-statement

formal-declaration-arguments:
  declaration
  formal-declaration-arguments , declaration

extern-function-declaration:
  external type-name extern-identifier ( optional-formal-declaration-arguments );

global-declaration:
  declaration ;

global-const-definition:
  constant type-name const-identifier = expression ;

declaration:
  type-name variable-identifier

type-name: one of
  int float bool matrix sequence string handle void

statement:
  expression-statement ;
  compound-statement
  selection-statement
  iteration-statement
  retn-statement

compound-statement:
  { optional-local-declaration-list optional-statement-list }

local-declaration-list:
  declaration ;
  local-declaration-list declaration ;

statement-list:
  statement
  statement-list statement

selection-statement:
  if ( expression ) statement
  if ( expression ) statement else statement

iteration-statement:
  while { expression } statement
  for ( optional-expression ; expression ; optional-expression ) statement

retn-statement:
  return optional-expression

primary-expression:
  identifier
  literal
  mat-literal
  seq-literal
  type-constructor
  ( expression )

mat-literal:
  [ optional-mat-row-list ; ]

mat-row-list:
  mat-expression-list
  mat-row-list ; mat-expression-list

expression-list:
  expression
  expression-list , expression

seq-literal:
  [ optional-seq-expression-list ]

type-constructor:
  new type-name ( optional-argument-expression-list )

postfix-expression:
  primary-expression
  mat-identifier [ row-index-expression , column-index-expression ]
  seq-identifier [ index-expression ]
  function-identifier ( optional-argument-expression-list )
  postfix-expression '

unary-expression:
  postfix-expression
  ! unary-expression
  - unary-expression
  length ( expression )
  cols ( expression )
  cast-name ( cast-expression )

cast-name: one of
  int_of_float float_of_int int_of_seq float_of_mat

power-expression:
  unary-expression
  power-expression ^ unary-expression
  power-expression .^ unary-expression

multiplicative-expression:
  power-expression
  multiplicative-expression * power-expression
  multiplicative-expression / power-expression
  multiplicative-expression % power-expression
  multiplicative-expression .* power-expression
  multiplicative-expression ./ power-expression
  multiplicative-expression .% power-expression

additive-expression:
  multiplicative-expression
  additive-expression + multiplicative-expression
  additive-expression - multiplicative-expression

colon-expression:
  additive-expression
  additive-expression : additive-expression : additive-expression
  additive-expression :: additive-expression
```

```
relational-expression:
  colon-expression:
  relational-expression > colon-expression
  relational-expression >= colon-expression
  relational-expression < colon-expression
  relational-expression <= colon-expression

equality-expression:
  relational-expression:
  equality-expression == relational-expression
  equality-expression != relational-expression

logical-AND-expression:
  equality-expression
  logical-AND-expression && equality-relation

logical-OR-expression:
  logical-AND-expression
  logical-OR-expression || logical-AND-expression

assignment-expression:
  logical-OR-expression
  unary-expression = assignment-expression

literal:
  integer-literal
  floating-literal
  boolean-literal
  str-literal
```

3.9 Library functions

This section describes the support functions coded in the MiniMat language itself that are included in several library files. These help implement all the matrix and sequence expressions and operators.

3.9.1 Expressions library

Functions coded in MiniMat language to help implement matrix literal and reference bracket expressions (included in `expressions.mm`):

Function	Description
int end(sequence)	return last index position of a sequence
int rows(matrix)	return number of rows of matrix
int size(matrix)	return capacity of a matrix
float float_of_string(string s)	convert string to float
int int_of_string(string s)	convert string to int
string string_of_int(int d)	convert int to string

string string_of_float(int f)	convert float to string
matrix mat_of_seq(sequence)	convert sequence to matrix
sequence seq_of_mat(matrix)	convert matrix to sequence
matrix vertcat(matrix, matrix)	concatenate columns of matrix vertically
matrix horzcat(matrix, matrix)	concatenate rows of matrix horizontally
matrix mselect(matrix, sequence, sequence)	subselect matrix by row and column sequences
matrix massign(matrix, sequence, sequence, matrix)	assign to submatrix by row and column sequences
matrix mat_select_seq(matrix, sequence)	select from submatrix by linear-index sequence
matrix mat_assign_seq(matrix, sequence, matrix)	assign to submatrix by linear-index sequence
sequence append(sequence, sequence)	concatenate sequences
sequence vselect(sequence, sequence)	select from subsequence by position sequence
sequence vassign(sequence, sequence, sequence)	assign to subsequence by position sequence
sequence stride(int, int, int)	construct sequence with colon stride pattern
void errorexit(string)	print error message and exit nicely

3.9.2 Operators library

Functions coded in MiniMat language to help implement matrix math and relational operators (included in `operators.mm`):

Function	Description
matrix madd(matrix, matrix)	implement matrix `+` infix operator: add matrices
matrix msub(matrix, matrix)	implement matrix `-` infix operator: subtract matrices
matrix mdotmul(matrix, matrix)	implement matrix `.*` infix operator: multiply element by element
matrix mdotdiv(matrix, matrix)	implement matrix `./` infix operator: divide element by element
matrix mdotrem(matrix, matrix)	implement matrix `./` infix operator: remainder element by element
matrix mdotpow(matrix, matrix)	implement matrix `.^` infix operator: raise each element to power of corresponding element
sequence mlt(matrix, matrix)	implement matrix `<` infix operator: return sequence of linear positions where element is less than corresponding element
sequence mle(matrix, matrix)	implement matrix `<=` infix operator: return sequence of linear positions where element is less or equal corresponding element
sequence mgt(matrix, matrix)	implement matrix `>` infix operator: return sequence of linear positions where element is greater than corresponding element
sequence mge(matrix, matrix)	implement matrix `>=` infix operator: return sequence of linear positions where element is greater or equal corresponding element
sequence meq(matrix, matrix)	implement matrix `==` infix operator: return sequence of linear positions where element is equal to corresponding element
sequence mne(matrix, matrix)	implement matrix `!=` infix operator: return sequence of linear positions where element is not equal to corresponding element
matrix mtransp(matrix)	implement matrix `'` postfix unary operator: return transpose of matrix
matrix mneg(matrix)	implement matrix `-` unary operator: negate every value of matrix

matrix mmul(matrix, matrix)	implement matrix `*` infix operator: multiply matrices
matrix mpow(matrix, int)	implement matrix `^` infix operator: multiply matrix by itself a specified number of times
matrix mdiv(matrix, matrix)	implement matrix `/` infix operator: coefficients of linear regression
matrix mrem(matrix, matrix)	implement matrix `%` infix operator: residuals from linear regression
sequence vadd(sequence, sequence)	implement sequence `+` infix operator: add sequence values
sequence vsub(sequence, sequence)	implement sequence `-` infix operator: subtract sequence values
sequence vmul(sequence, sequence)	implement sequence `*` infix operator: multply sequence values
sequence vdiv(sequence, sequence)	implement sequence `/` infix operator: divide sequence values
sequence vrem(sequence, sequence)	implement sequence `%` infix operator: remainder of sequence values
sequence vneg(sequence)	implement sequence `-` unary operator: negate sequence values
bool stringeq(string, string)	implement string `==` infix operator: return true if two strings have same characters
bool stringne(string, string)	implement string `!=` infix operator: return true if two strings do not have same characters
bool stringge(string, string)	implement string `>=` infix operator: return true if string is equal or after another
bool stringgt(string, string)	implement string `>` infix operator: return true if string is after another
bool stringle(string, string)	implement string `<=` infix operator: return true if string is equal or before another
bool stringlt(string, string)	implement string `<` infix operator: return true if string is before another

3.9.3 Functions library

Functions coded in MiniMat language to implement matrix math functions (included in `functions.mm`).

Function	Description
float fabs(float)	return absolute value of float
float exp(float)	return exponential of float
float log(float)	return log of float
float pow(float, float)	return float raised to power of float
float sqrt(float)	return square root value of float
matrix mexp(matrix)	return matrix of exponential values of each element
matrix mlog(matrix)	return matrix of log value of each element
matrix mabs(matrix)	return matrix of absolute values of each element
matrix eye(int)	return identity matrix
matrix diag(matrix)	return diagonal of matrix
matrix ones(int, int)	return matrix of ones
matrix reshape(matrix, int, int)	reshape matrix to given dimensions
float sum(matrix)	return sum of elements of matrix
float mean(matrix)	return mean value of elements of matrix

float norm(matrix)	return euclidean norm value of elements of matrix
float min(matrix)	return minimum value in matrix
float max(matrix)	return maximum value in matrix
float det(matrix)	return determinant value of matrix
matrix cofactor(matrix)	return cofactor of matrix
matrix tril(matrix, int)	return lower triangular matrix
matrix triu(matrix, int)	return upper triangular matrix
matrix adjoint(matrix)	return adjoint of matrix
matrix inv(matrix)	return inverse of matrix
matrix regress(matrix, matrix)	return fitted values from linear regression

3.9.4 Input/Output library

Functions coded in MiniMat language to implement input/output (included in `io.mm`).

Function	Description
void println()	print newline to standard output
void printint(int)	print an int to standard output
void printbool(bool)	print a bool to standard output
void printfloat(float)	print a float to standard output
void printstring(string)	print a string to standard output
void printhandle(handle)	print address value in handle to standard output
void printdims(matrix)	print row and column dimensions of matrix
void printseq(sequence)	print values of sequence
void printmat(matrix)	print values of matrix
string next()	get next string from standard input
float nextfloat()	get next float from standard input
int nextint()	get next int from standard intput

Additionally, the following output functions are built-in, and help provide compatibility with MicroC and its test suite.

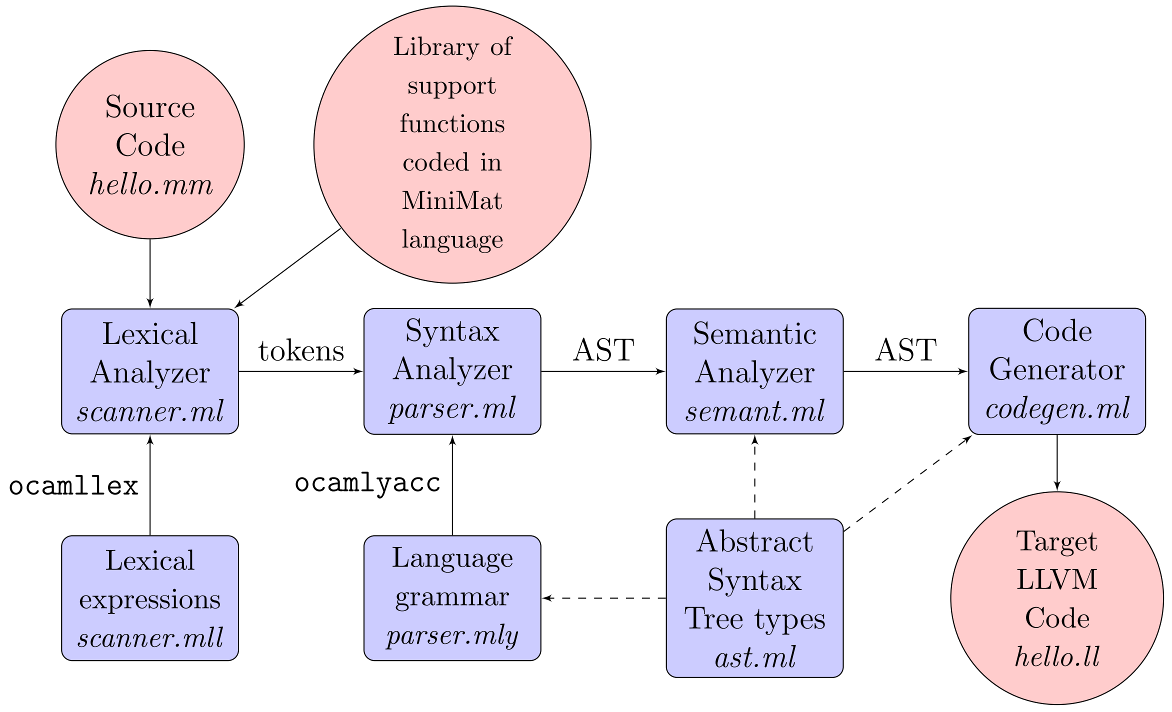
Function	Description
print(int)	print an integer and newline
printb(bool)	print a boolean and newline
printf(string,...)	print a variable list of arguments according to format string

4 Architectural Design

4.1 Major Components

The compiler front end components (lexer, parser, semantic analysis and intermediate code generation) are written in OCaml, OCamllex, OCamlyacc and OCaml Llvm, and translates source language into LLVM IR target code. The target file can then be directly executed with the `lli` interpreter, or compiled with `llc` to native assembly language.

Figure 1: Block diagram of major components of MiniMat language translator



Source Code *hello.mm*
Library of support functions coded in MiniMat language
Lexical Analyzer *scanner.ml*
tokens
Syntax Analyzer *parser.ml*
AST
Semantic Analyzer *semant.ml*
AST
Code Generator *codegen.ml*
ocamllex
ocamlyacc
Lexical expressions *scanner.mll*
Language grammar *parser.mly*
Abstract Syntax Tree types *ast.ml*
Target LLVM Code *hello.ll*

4.2 Interfaces Between Components

Lexical rules for token scanning were specified in *scanner.mll*, then `ocamllex` generated the OCaml lexical analyzer program. Syntax rules for the grammar were specified in *parser.mly*, then `ocamlyacc` generated the OCaml parser program. The code generator program *codegen.ml* uses the OCaml LLVM bindings. The entry point for the compiler is an OCaml program *minimat.ml* which orchestrates the interface between the components: concatenating the input source files (i.e. the program and the support library source files located in `./include/*.mm`) for lexical analysis, sending the stream of tokens for parsing, passing the AST to be type-checked, and finally outputting generated code in the target LLVM IR.

4.2.1 Interfacing with External C Functions

In the MiniMat language, external functions can be used upon declaring their prototypes following the `external` keyword. Hence the compiler did not need to individually build-in calls of any primitive math library system functions such as `pow(), log(), exp()` and `fabs()`; they are declared only as and when needed in a program or library source file coded in MiniMat itself. Similarly, external library routines such as the public domain gnuplot[5] to visualize computations can be used.

If external C functions are called, the LLVM IR generated code should not be executed by the interpreter `lli`; instead it is compiled with `llc` into assembly language, then passed through a native assembler and, along with the external C library object files, a linker such as `gcc` to generate a native executable. To load the system math library, link with the option `gcc -lm`. As described in the tutorial section, a convenient shell script `mmc` has been provided to wrap all these compilation tasks to use the gnuplot external C library.

4.3 Language Design and Implementation Choices

4.3.1 Internal Representation of a Matrix

A matrix data type is represented in the target LLVM language as an array of floats of variable size (with a `[0 x double]` struct template) prepended with a 16-byte header that identifies the actual storage size, #cols and #rows. Hence the size of a matrix associated with an identifier can change, and is tracked automatically along with the object contents.

4.3.2 Constructing Bracket Expressions

Matrices are constructed and augmented with "matlab-like" bracket expressions. Matrix literals, enclosed by brackets, are parsed into lists (i.e. rows) of lists (i.e. columns) of stuff (i.e. floats or smaller matrix expressions), then the compiler's code generation component calls `vertcat()` and `horzcat()`, which are standalone helper functions coded in MiniMat language, repeatedly on every element column-by-column and row-by-row (i.e. nested "fold left" iterations) to build up its LLVM storage representation.

4.3.3 Stack versus Heap Storage

Matrix (and sequence and string) objects, that are local variables or temporary computations, are allocated from the stack which simplifies memory management but may be lost when returning functions pop their call frame. For functions to return large data types such as matrices, the compiler takes care of temporarily copying to a block of heap memory, then back to the stack of the caller. The only exception is global variables: when a large data object is assigned to a global identifier, it is first copied to a block allocated from the heap, so that it persists across function calls. Minimat does not explicitly implement back-end garbage collection of heap memory (perhaps beyond the scope of this project), so memory

[5]To use the gnuplot external functions, you must have gnuplot installed and working on your machine, accessible from the user account you are using. The sample test program also requires gnuplot to have been compiled with PNG support. This should be the case if you are using a pre-packaged gnuplot under Linux.

leakage may result as global matrix (or sequence or string) identifiers are reassigned new storage.

4.3.4 One-dimensional Matrices

One design issue we wrestled with was whether to implement a one-dimensional floating matrix (i.e. vector). However, one-dimensional floating matrices may introduce undesirable behavior. It is not clear whether it is a row ($1 \times n$) or column ($n \times 1$) matrix that conforms for operations with other matrices. Such a matrix has to be arbitrarily "shaped" (i.e. specify which of number of rows or columns equals 1) before, say, multiplying with two-dimensional matrices; but it could be important to know whether that actually should have been a $n \times 1$ or $1 \times n$ matrix passed in, and possibly disallow matrix multiplication or select a different calculation. Nevertheless, MiniMat does implement a one-dimensional list of integers as the `sequence` data type. However, without explicit type conversion, a `sequence` operand is not accepted by matrix operators and functions. Instead, this data type is most useful to serve as index references for simultaneously retrieving or placing multiple values in a matrix by specifying a sequence of row and/or column positions.

4.3.5 Matrix Transpose as Postfix Operator

The matrix transpose operator `'` is a postfix expression: though ugly from a language design perspective, it resembles familiar matrix notation which is a key goal of MiniMat syntax. But user programs can prefer to use its helper function directly: `mtransp(A)` instead of `A'`.

4.4 Development Environment

Operating System:	Ubuntu 14.04
Development language:	OCaml 4.02.3 (and OCamllex, OCamlyacc and OCaml Llvm bindings)
Target language:	LLVM version 3.4
Editor:	GNU Emacs 24.3 (caml mode)
External graphics library:	gnuplot_i C API (in public domain, by N. Devillard)
Linker:	gcc 4.8

5 Sample Programs

5.1 Source Program I: Matrix Inverse and Linear Regression

This sample program demonstrates MiniMat code for:

- matrix bracket expression statements: define and augment matrices, such as inserting a column of ones into a matrix block;
- implementing a matrix infix operator, such as the matrix division `/` operator, by defining a helper routine that is bound to the operator which can also be called as a standalone function;
- defining matrix math functions such as `inv` and `det` for matrix inverse and determinant respectively – the latter function is defined recursively; and
- build up more abstract formulas such as a (one-line) statistical program to identify outlier observations from fitting a linear regression.

There is no standard mathematical definition of matrix division `/`, so we shall provide our own by coding it up as a helper function named `mdiv` (which is bound to the matrix infix operator `/`) in the MiniMat language which shall be incorporated into its operators library source file `operators.mm`. We define this operator to return the least square regression coefficients $Y \; / \; X = \beta = (X'X)^{-1}X'Y$, so that with scalar operands for example, the quotient β exactly satisfies $Y = \beta X$, though only approximately satisfies (but with minimum squared error) $Y \simeq \beta X$ with matrix operands. The regression formula also requires coding a new function, named `inv`, to compute a matrix inverse.

The source language program is listed below, while its translated LLVM code is presented in the appendix.

Listing 3: Defining and using functions in MiniMat – regression.mm

```
/**/
/*------------------------------------------------------------
  MDIV -- define function for matrix divide infix operator "/"
------------------------------------------------------------*/
matrix mdiv(matrix y, matrix x) {
  checkmatrows(y,x);
  return inv(x' * x) * (x' * y);
}
/**/
/**/
/*------------------------------------------------------------
  DET -- computes determinant by recursively expanding minors
------------------------------------------------------------*/
float det(matrix a) {
  matrix det;
  int i;
  int j;
  int j1;
  int j2;
  matrix m;
  float tmp;
  checkmatsquare(a);
  if (rows(a) == 1) det = a[0, 0];
  else if (rows(a) == 2) det = a[0, 0] * a[1, 1] - a[0, 1] * a[1, 0];
  else {
    det = [0.0;];
    for (j1 = 0; j1 < cols(a); j1 = j1 + 1) {
      m = new matrix(rows(a) - 1, cols(a) - 1);
      for (i = 1; i < rows(a); i = i + 1) {
        j2 = 0;
        for (j = 0; j < cols(a); j = j + 1) {
           if (j != j1) {
             m[i-1, j2] = a[i, j];
             j2 = j2 + 1;
           }
        }
      }
      det = det + [(-1.0 ^ (float_of_int(j1) + 2.0));] * a[0, j1] * [det(m);];
    }
  }
  return float_of_mat(det);
}

/*-------------------------------------------------------------
  COFACTOR -- returns cofactor of a matrix
--------------------------------------------------------------*/
matrix cofactor(matrix a) {
  int i;
  int j;
  int ii;
  int jj;
  int i1;
  int j1;
  float det;
  matrix c;
  int n;
  matrix b;
  checkmatsquare(a);
  n = rows(a);
  b = new matrix(n, n);
  c = new matrix(n-1, n-1);
  for (j = 0; j < n; j = j + 1) {
    for (i = 0; i < n; i = i + 1) {
      i1 = 0;
      for (ii = 0; ii < n; ii = ii + 1) {
        if (ii != i) {
          j1 = 0;
          for (jj = 0; jj < n; jj = jj + 1) {
            if (jj != j) {
              c[i1, j1] = a[ii, jj];
              j1 = j1 + 1;
            }
          }
          i1 = i1 + 1;
        }
      }
      b[i, j] = [(-1.0 ^ (float_of_int(i+j)+2.0)) * det(c);];
    }
  }
  return b;
}

/*---------------------------------------------------------------
  INV -- returns inverse of matrix
--------------------------------------------------------------*/
matrix inv(matrix a) { return cofactor(a)' ./ [det(a);]; }
/**/
```

```
int main() {
  matrix y;
  matrix x;
  matrix xx;
  sequence outliers;

  /* create demonstration data set */
  y = [2.0; 0.5; 1.5; 5.0; 7.0; 7.0;];
  x = [1.0,2.0; 2.0,2.0; 3.0,3.0; 4.0,3.0; 5.0,5.0; 6.0,6.0;];

  /* insert column of ones */
  xx = new matrix(rows(x), 1) + [1.0;];
  x = [xx, x;];

  /* check accuracy of inv() */
  xx = x' * x;
  printmat(inv(xx));
  printmat(inv(xx) * xx);

  /* compute outliers */
  outliers = y - x*(y/x) < [-1.0;];
  printmat([y, x;]);  /* print matrix of y and x side-by-side */
  printstring("Number of outliers: ");
  printint(length(outliers));
  println();
}
```

5.2 Source Program II: Logistic Regression

This next sample program demonstrates how to implement a significant statistical algorithm in MiniMat making full use of its matrix notation and procedural statements: estimating a logistic regression model with the Newton-Raphson method. Numerical termination thresholds – the maximum norm of changes in estimates, and the number of iterations to run – are naturally defined as immutable constants at the top of the program. To help visualize (which is a common task in statistical programming) the estimates iteratively converging, we link and call an external C library, *gnuplot*, which uses the language's `external` modifier to declare external function prototypes, `new string` constructor to create text labels and commands, and `handle` type for a pointer required by gnuplot to access its session object. The graphical output generated is presented in Figure 2 below.

The source language program is listed below, while its translated LLVM code is presented in the appendix.

Listing 4: Coding a statistical algorithm in MiniMat – logistic.mm

```
/*
  logistic regression by iterated least squares/Newton-Raphson method.
  example adapted from http://strijov.com/sources/demo_logistic_regression.php
*/

constant float MAXNORM = 0.000001;   /* threshold for convergence */
constant int   MAXITER = 8;          /* max number of iterations */

/* reweight each row of data matrix by column of weights */
matrix mweighted(matrix x, matrix w) {
  matrix y;
  int r;
  checkmatrows(x, w);
  y = new matrix(rows(x),cols(x));
  for(r = 0; r < rows(x); r = r + 1) {
    y[r, 0::cols(y)-1] = x[r, 0::cols(y)-1] * w[r, 0];
  }
  return y;
}

/* computes logistic regression from labels and vars (exclude intercept) */
matrix logistic(matrix labels, matrix vars, string s) {
  int iter;
  float delta;
  matrix beta;
  matrix prev;
  matrix X;
  matrix z;
  matrix p;
  matrix w;
  matrix u;
  matrix x;
  handle g;

  /* open a gnuplot session to plot fit iterations */
  g = gnuplot_init();      /* pointer to gnuplot session object */

  if (s == "") gnuplot_cmd(g, "set terminal xterm");
  else gnuplot_set_png(g, s);       /* select terminal or png file output */

  gnuplot_cmd(g, "set multiplot");  /* set axes, labels and plot styles  */
  gnuplot_set_yrange(g, labels);
  gnuplot_set_xrange(g, vars);
  gnuplot_set_ylabel(g, "estimated logistic probability");
  gnuplot_set_xlabel(g, "x");
  gnuplot_cmd(g, "set key top left Left reverse");
  gnuplot_setstyle(g, "linespoints");

  /* initialize parameters */
  X = [ones(rows(vars),1), vars;];
  beta = new matrix(cols(X),1);
  beta[0, 0] = [log(mean(labels) / (1.0 - mean(labels)));];
  delta = MAXNORM;

  /* iterate till  max iter, or little change in estimates */
  for(iter = 1; iter <= MAXITER && delta >= MAXNORM; iter = iter + 1) {
    prev = beta;
    z = X * beta;                 /* update probability estimates */
    p = [1.0;] ./ ([1.0;] + mexp(-z));
    w = p .* ([1.0;] - p);        /* update weights */
    u = z + ((labels - p) ./ w);  /* reweight data */
    x = mweighted(X, w .^ [0.5;]);
    u = mweighted(u, w .^ [0.5;]);
    beta = inv(x' * x) * (x' * u); /* update coeffs with reweighted data */
    delta = norm(beta - prev);     /* check magnitude of parameter changes */

    printstring("iter");           /* display iteration */
    printint(iter);
    printstring(":");
    printfloat(delta);
    println();
    gnuplot_plot_xy(g, vars, p, rows(p), string_of_int(iter));
  }

  /* gnuplot initial and final data points */
  p = [1.0;] ./ ([1.0;] + mexp(-X * beta));   /* final estimates */
  gnuplot_plot_xy(g, vars, p, rows(vars), "recovered data");
  gnuplot_setstyle(g, "points");
  gnuplot_plot_xy(g, vars, labels, rows(vars), "initial data");
  gnuplot_close(g);
  return beta;
}

int main() {
  matrix x;
  matrix y;

  /* create demonstration data set */
  x = [mat_of_seq(-8::1), mat_of_seq(2::11);]';
  y = [new matrix(9, 1); 1.0; 0.0; ones(9, 1);];

  /* display logistic regression results and save graph as PNG */
  printmat(logistic(y, x, "logistic.png"));
}
```

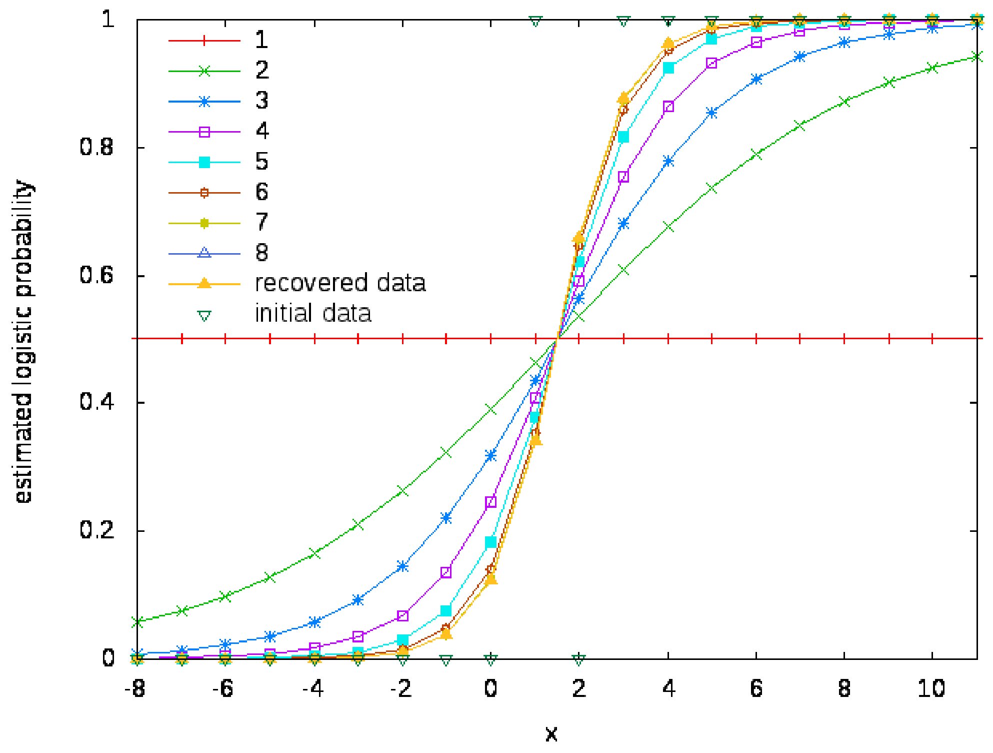
```
/**/
/* Declare external GNUPLOT C API -- for visualizing plots */
external handle gnuplot_init();
external void gnuplot_cmd(handle g, string c);
external void gnuplot_plot_equation(handle g, string c,string s);
external void gnuplot_close(handle g);
external void gnuplot_plot_xy(handle g, matrix x, matrix y, int n, string s);
external void gnuplot_setstyle(handle g, string s);
  /* lines points linespoints impulses dots steps errorbars boxes */
external void gnuplot_resetplot(handle g);
external void gnuplot_set_xlabel(handle g, string s);
external void gnuplot_set_ylabel(handle g, string s);

/* sets output to a PNG picture file */
void gnuplot_set_png(handle g, string f) {
  gnuplot_cmd(g, "set terminal png");
  gnuplot_cmd(g, new string("set output \"%s\"", f));
}

/* sets yrange of plot from min and max values of data set */
void gnuplot_set_yrange(handle g, matrix y) {
  gnuplot_cmd(g, new string("set yrange [%g:%g]", min(y), max(y)));
}

/* sets xrange of plot from min and max values of data set */
void gnuplot_set_xrange(handle g, matrix x) {
  gnuplot_cmd(g, new string("set xrange [%g:%g]", min(x), max(x)));
}
/**/
```

Figure 2: Visualizing computations by calling externally-declared gnuplot functions



6 Test Cases

As I developed the compiler, I wrote tests every time that I implemented a new feature to verify that it works. For each new set of features, I wrote a minimum of two test programs: One intended to pass, and at least one intended not to. Some of these “failing” cases compiled and executed, but detected run-time errors and displayed diagnostic messages, such as invalid dimensions and/or index position operands of matrix functions and operators. In subsequent development, I ran a shell script to automatically rerun the accumulated inventory of test programs and compare (with the unix `diff` utility) that all the output remains exactly the same. My `testmm.sh` script[6] draws from `testall.sh` included in the *MicroC* distribution, and in fact that latter script is also run as part of the testing suite since MiniMat implements a superset of the MicroC language. Overall, test cases were created to check:

- Types – floating and string literals, declarations and operators: `test-type-float` `test-type-string` `fail-type-float`
- Matrix expressions – matrix bracket expressions to construct matrix literal; augment and reshape matrices; and retrieve or place values in multiple positions in a matrix: `test-expr-mat` `fail-expr-mat1` `fail-expr-mat2`
- Sequence expressions – similar tests for `sequence` type: `test-expr-vec` `fail-expr-vec1` `fail-expr-vec2`
- Sizeof and type-cast operators: `test-expr-sizeof` `test-type-cast`
- Matrix and sequence operators – all mathematical and relational operators: `test-op-mat` `test-op-vec`
- Matrix sizes are checked for bounds and consistency at run-time: `test-expr-mat2` `test-expr-mat3` `test-mat-mul` `test-mat-augment` `test-mat-assign` `test-seq-assign` `test-seq-select` `test-mat-add`
- Matrix functions – extensive library of matrix math functions: `test-func-mat`
- Global variables and constants – declaration and definition of literals of all data types, and scope: `text-constant-decl` `test-globals` `test-var-init` `fail-constant-decl` `fail-constant-decl2` `fail-constant-decl3`
- Superset of MicroC – iteration and selection statements; integer and boolean declarations, operators and functions: `testall.sh`

6.1 Code Listings of the Test Suite

The test suite programs are in `./tests/*.mm`.

[6] Requires `lli`.

Listing 5: Code listings of the test suite

```
::::::::::::::
test-constant-decl.mm
::::::::::::::
/* test-constant-decl: definitions and scope */
constant int j      = 2 - 3;            /* definitions: all types */
constant float k    = -2.3;
constant bool l     = !true;
constant string s   = "hello, world";
constant matrix a   = [1.0; 2.0;]';
constant sequence x = new sequence(2);

void printconstants() {                 /* global scope */
  printint(j);
  printfloat(k);
  printbool(l);
  printstring(s);
  println();
  printmat(a);
  printseq(x);
}

void printlocals() {    /* scope of constants suspended by local decls */
  int j;
  float k;
  bool l;
  string s;
  matrix a;
  int m;
  j = 99;
  k = 999.0;
  l = true;
  s = "goodbye";
  a = [99.9, 999.99, 9999.0;];
  m = 9999;
  printint(j);
  printfloat(k);
  printbool(l);
  printstring(s);
  println();
  printmat(a);
  printseq(x);
}

int main() {
  printconstants();
  printlocals();
}
::::::::::::::
test-expr-mat.mm
::::::::::::::
/* test-expr-mat: matrix assignment, subselect, definition */
constant matrix x = [-99.0; -9.9;];      /* matrix constant definition */
```

```
int main() {
  matrix a;
  matrix b;
  matrix c;
  sequence u;
  sequence v;
  a = [  1.1,  2.1,  3.1,  4.1;
         5.2,  6.2,  7.2,  8.2; ];      /* construct matrix literal */
  b = [ 11.1, 12.1, 13.1, 14.1;
        15.2, 16.2, 17.2, 18.2; ];
  printmat(a);
  printmat([;]);                        /* empty matrix */
  printmat([a; b;]);                    /* horzcat matrix */
  printmat([a', b';]);                  /* vertcat matrix */
  printmat([a, [;];]);                  /* one matrix can be empty */
  printmat([[;], b;]);
  printmat([a; [;];]);
  printmat([[;]; b;]);
  printmat(a[1, 2]);                    /* subselect matrix */
  u = 1::1;
  v = 1:2:3;
  printmat(a[u, v]);
  b[1, 3:-2:1] = a[0, v];               /* subassign matrix */
  printmat(b);
  b = a;                                /* assignment (copies values) */
  b[0,2::3] = x';
  printmat(a);
  printmat(b);
  printmat(a[1,2::3] = b[0,2::3] = [-99.9, -88.8;]);  /* chained assignment */
  printmat(a);
  printmat(b);
  printmat(a[1,0] = b[0,1] = [-77.7;]);
  printmat(a);
  printmat(b);
}
::::::::::::::
test-expr-mat2.mm
::::::::::::::
/* test-expr-mat2: matrix assignment, subselect, definition */
int main() {
  matrix a;
  a = [1.0; 2.0;];    /* 2 x 1 matrix */
  a = [a, a';];       /* cannot horzcat matrices with different rows */
}
::::::::::::::
test-expr-mat3.mm
::::::::::::::
/* test-expr-mat3: matrix assignment, subselect, definition */
int main() {
  matrix a;
  a = [1.0; 2.0;];    /* 2 x 1 matrix */
  a = [a; a';];       /* cannot vertcat matrices with different columns */
}
::::::::::::::
```

```
test-expr-sizeof.mm
::::::::::::::
/* test-expr-sizeof: matrix and vector sizeof functions */
int main() {
  matrix u;
  sequence v;
  u = [1.0, 2.0, 3.0, 4.0; 11.0, 12.0, 13.0, 14.0;];
  v = [1, 2, 3, 4, 5, 6];
  printint(length(v));   /* length operator */
  printint(end(v));      /* sequence end function */
  printint(size(u));     /* matrix size function */
  printint(cols(u));     /* matrix cols operator */
  printint(rows(u));     /* matrix rows function */
  println();
}
::::::::::::::
test-expr-vec.mm
::::::::::::::
/* test-expr-vec: vector assignment, subselect, definition, colon expression
   */
constant sequence x = [-99];     /* sequence constant definition */
int main() {
  sequence u;
  sequence v;
  sequence w;
  v = [1, 2, 3, 4];              /* construct sequence literal */
  printseq(v);
  printseq([]);                  /* empty sequence */
  v = [10, v, v, 11];            /* augment sequence */
  printseq(v);
  v = [101::105, 110:-1:106];    /* colon expression */
  printseq(v);
  printseq(v[2]);                /* subselect vector */
  u = [2, 5, 6];
  printseq(v[u]);
  v[0] = [200];                  /* subassign vector */
  printseq(v);
  v = u;                         /* assignment (copy values) */
  v[0] = [x];
  printseq(u);
  printseq(v);
  printseq(u[1] = v[1] = [99]);  /* chain assignment */
  printseq(u = v = [88, 888]);
}
::::::::::::::
test-func-mat.mm
::::::::::::::
/* test-func-mat: library of matrix functions */
int main() {
  matrix u;
  u = [1.0, 2.0, -3.0; 4.0, -5.0, 6.0; 7.0, 8.0, -9.0;];
  printmat(mabs(u));     /* absolute values */
  printmat(mexp(u));     /* exponent values */
  printmat(mlog(u));     /* log values */
  printmat(eye(2));      /* identity matrix */
  printmat(diag(u));     /* diagonal of matrix */
  printmat(ones(2,4));   /* matrix of ones */
  printfloat(sum(u));    /* sum of matrix values */
  printfloat(mean(u));   /* mean of matrix values */
  printfloat(norm(u));   /* euclidean norm of matrix values */
  printfloat(min(u));    /* minimum of matrix values */
  printfloat(max(u));    /* maximum of matrix values */
  printfloat(det(u));    /* determinant of matrix */
  printmat(triu(u,0));   /* upper triangular matrix */
  printmat(tril(u,0));   /* lower triangular matrix */
  printmat(adjoint(u));  /* adjoint of matrix */
  printmat(inv(u));      /* inverse of matrix */
  printmat(inv(u) * u);  /* check inverse */
}
::::::::::::::
test-globals.mm
::::::::::::::
/* test-globals: global variables identifier names, definitions and scope */
int      the_int;       /* declare global variables: all types */
float    the_float;
bool     the_bool;
string   the_string;
matrix   the_matrix;
sequence the_sequence;

constant int      the_int      = 99;      /* constants are suspended  */
constant float    the_float    = -999.99;
constant bool     the_bool     = !true;
constant string   the_string   = " constant ";
constant matrix   the_matrix   = [999.99; 999.99;];
constant sequence the_sequence = [99, 99];

int      the_int()      { return  55; }    /* function names not interfere */
float    the_float()    { return  55.55; }
bool     the_bool()     { return  false; }
string   the_string()   { return  " function "; }
matrix   the_matrix()   { return  [555.55, 555.55;]; }
sequence the_sequence() { return  [55, 55]; }

void printglobals() {           /* scope of global variables */
  printint(the_int);
  printfloat(the_float);
  printbool(the_bool);
  printf(the_string);
  println();
  printmat(the_matrix);
  printseq(the_sequence);
}

void printfunctions() {         /* function names do not interfere */
  printint(the_int());
  printfloat(the_float());
  printbool(the_bool());
  printf(the_string());
  println();
  printmat(the_matrix());
  printseq(the_sequence());
}

void printlocals() {            /* scope suspended by local declarations */
  int      the_int;
  float    the_float;
  bool     the_bool;
  string   the_string;
  matrix   the_matrix;
  sequence the_sequence;
  the_int      = 88;
  the_float    = 88.88;
  the_bool     = true;
  the_string   = " local ";
  the_matrix   = [888.88, 888.88;];
  the_sequence = [88, 88];
  printint(the_int);
  printfloat(the_float);
  printbool(the_bool);
  printf(the_string);
  println();
  printmat(the_matrix);
  printseq(the_sequence);
}

int changeglobals() {
  the_int      = 44;            /* change values of globals, will persist */
  the_float    = 44.44;
  the_bool     = false;
  the_string   = " changed ";
  the_matrix   = [444.44, 444.44;];
  the_sequence = [44, 44];
}

int main() {
  the_int      = 77;
  the_float    = 77.77;
  the_bool     = true;
  the_string   = " global ";
  the_matrix   = [777.77, 777.77;];
  the_sequence = [77, 77];
  printglobals();                 /* print globals */
  printlocals();                  /* print locals, suspends globals */
  printfunctions();               /* print functions, do not interfere */
  changeglobals();                /* change values of globals */
  printglobals();                 /* reprint globals with changed values */
}
::::::::::::::
test-mat-add.mm
::::::::::::::
/* check-mat-add -- not same size */
```

```
int main() {
  matrix a;
  a = new matrix(4,2);
  printmat(a + a');      /* cannot add matrices of different sizes */
}

::::::::::::::
test-mat-assign.mm
::::::::::::::
/* check-mat-assign -- index out of bounds */
int main() {
  matrix a;
  a = new matrix(2, 2);
  a[2, 2] = [1.0;];      /* cannot assign to matrix out of bounds */
}

::::::::::::::
test-mat-augment.mm
::::::::::::::
/* check-mat-augment -- adjacent matrix rows do  not have same #columns */
int main() {
  matrix a;
  a = [1.0, 2.0, 3.0;];
  [a; a';];              /* cannot augment matrices with different columns */
}

::::::::::::::
test-mat-mul.mm
::::::::::::::
/* test-op-matmul: matrix mult checks operands are conformable */
int main() {
  matrix u;
  u = [1.0, 2.0, 3.0, 4.0;
      11.0, 12.0, 13.0, 14.0;];
  printmat(u * u);     /* matrix sizes do not conform for multiplication */
}
::::::::::::::
test-op-mat.mm
::::::::::::::
/* test-op-mat: matrix operators */
int main() {
  matrix u;
  matrix v;
  matrix w;
  matrix y;
  matrix x;
  u = [1.0, 2.0, 3.0, 4.0; 11.0, 12.0, 13.0, 14.0;];
  v = u;
  v[0,0] = [1000.0;];
  printmat(u);
  printmat(v);
  printmat(u + v);              /* arithmetic */
  printmat(v + u);
  printmat(u - v);
  printmat(v - u);
  printmat([1.0, 2.0; 3.0, 4.0;] ^ 3);
  printmat(-v);                /* unary */
  printmat(v');
  printmat(v + [1000.0;]);     /* arithmetic with scalar matrix*/
  printmat([2000.0;] - v);
  printmat(u .* u);            /* elemental */
  printmat(v ./ u);
  printmat(v .% u);
  printmat(v .^ [3.0;]);
  printseq(u < v);             /* relational */
  printseq(u <= v);
  printseq(u > v);
  printseq(u >= v);
  printseq(u == v);
  printseq(u != v);

  printmat(u * u');                        /* matrix multiplication */
  y = [2.0; 0.5; 1.5; 5.0; 7.0; 7.0;];    /* matrix regression */
  x = [1.0,2.0; 2.0,2.0; 3.0,3.0; 4.0,3.0; 5.0,5.0; 6.0,6.0;];
  x = [new matrix(rows(x), 1) + [1.0;], x;];
  printmat(y/x);
  printmat(y%x);
}
::::::::::::::
test-op-vec.mm
::::::::::::::
/* test-op-vec: vector operators */
int main() {
  sequence v;
  sequence u;
  sequence w;
  v = 1::5;               /* colon expression */
  u = 100:2:108;
  printseq(u + v);        /* arithmetic */
  printseq(v + u);
  printseq(u - v);
  printseq(v - u);
  printseq(u * v);
  printseq(v * u);
  printseq(u / v);
  printseq(v / u);
  printseq(u % v);
  printseq(-v);
  printseq(v + [1000]);   /* arithmetic with scalar sequence */
  printseq([2000] + v);
  printseq([2000] / v);
}
::::::::::::::
test-seq-assign.mm
::::::::::::::
/* check-seq-assign -- sequence index out of bounds*/
int main() {
  sequence a;
  a = new sequence(2);
  a[2] = [1];          /* cannot assign to sequence out of bounds */
}

::::::::::::::
test-seq-select.mm
::::::::::::::
/* check-seq-select -- sequence index out of bounds*/
int main() {
  sequence a;
  a = new sequence(2);
  printseq(a[2]);          /* cannot reference sequence out of bounds */
}

::::::::::::::
test-type-cast.mm
::::::::::::::
/* test-type-cast: type conversion operators */
int main() {
  printint(int_of_float(3.6));                    /* float to int */
  printfloat(float_of_int(3));                    /* int to float */
  printint(int_of_string("3.7"));                 /* string to int */
  printstring(string_of_int(3));                  /* int to string */
  printstring(string_of_float(5.5));              /* float to string */
  printfloat(float_of_string("5.5"));             /* string to float */
  printint(int_of_seq([7]));                      /* scalar sequence to int */
  printfloat(float_of_mat([7.7;]));               /* scalar matrix to float */
  println();
  printmat(mat_of_seq([1, 2, 3, 4]));             /* sequence to matrix */
  printseq(seq_of_mat([1.0, 2.0, 3.0, 4.0;])); /* matrix to sequence */
}
::::::::::::::
test-type-float.mm
::::::::::::::
/* test-type-float: float literals, operators and relationals */
float addme(float x) {   /* float function definition and argument */
  return x + 39.5;
}
int main() {
  float a;
  float b;
  float c;
  a = -2.3;                /* literals */
  b = 4.6 + 3.0;
  printfloat(-a);          /* arithmetic operators */
  printfloat(a);
  printfloat(a + b);
  printfloat(a - 10.9);
  printfloat(5.0 * b);
  printfloat(a / b);
  printfloat(a % 2.0);
  printfloat(a ^ 3.0);
  printb(a == -2.3);        /* relationals */
  printb(a == b);
  printb(a != b);
  printb(a >= b);
  printb(a > b);
  printb(a <= b);
  printb(a < b);
  printfloat(addme(2.5)); /* function call */
  println();
}
::::::::::::::
test-type-string.mm
::::::::::::::
/* test-type-string: string literals, relationals and functions */
string gets(string prompt) {     /* string function definition and argument */
  printstring(prompt);
  return("goodbye");
}
int main() {
  string s;
  string t;
  s = "hello, world\n";          /* literals */
  printstring(s);
  t = s;                         /* assignment */
  s = gets("Enter the word \"goodbye\" without quotes: ");
  printstring(s);
  printstring(t);
  printbool(s == "goodbye");     /* literals */
  printbool(s != "goodbye");
  printint(int_of_string("0500"));      /* conversions */
  printfloat(float_of_string("1.0"));
  println();
}

::::::::::::::
test-var-init.mm
::::::::::::::
/* test-var-init: variables are initialized to default values when declared */
int i;          /* global variable declaration: all types */
float f;
bool b;
matrix m;
sequence v;
string s;

void printlocals() {
  int i;
  float f;
  bool b;
  matrix m;
  sequence v;
  string s;
  printmat(m);  /* print locally-declared variables: no seg faults */
  printseq(v);
  printf("\"%s\" ",s);
  printint(i);
  printfloat(f);
  printbool(b);
  println();
}

int main() {
  printmat(m);    /* print globally-declared variables: no seg faults */
  printseq(v);
  printf("\"%s\" ",s);
  printint(i);
  printfloat(f);
  printbool(b);
  println();
  printlocals();
}
::::::::::::::
fail-constant-decl.mm
::::::::::::::
/* fail-constant-decl: definitions and scope */
constant int m = 3;
int main() {
  m = 2;          /* cannot assign to or modify constant */
}
::::::::::::::
fail-constant-decl2.mm
::::::::::::::
/* fail-constant-decl2: constants can only be defined with literal values */
float b;
constant matrix a = [1.0, b;];  /* cannot define constant with identifiers */
int main() {
  b = [2.0, 3.0, 4.0;];
  printmat(b);
  printmat(a);
}
::::::::::::::
fail-constant-decl3.mm
::::::::::::::
/* fail-constant-decl3: duplicate constant definition */
constant int j = 2 - 3;
constant int j = 1;        /* duplicate constant identifier */
int main() {}
::::::::::::::
fail-expr-mat1.mm
::::::::::::::
/* fail-expr-mat1: matrix assignment, subselect, definition */
int main() {
  matrix a;
  a = [1000;];     /* matrix literal cannot comprise ints */
}
::::::::::::::
fail-expr-mat2.mm
::::::::::::::
/* fail-expr-mat2: matrix assignment, subselect, definition */
int main() {
  matrix a;
  a = [1000.0];     /* matrix rows must be terminated by semi-colon */
}
::::::::::::::
fail-expr-vec1.mm
::::::::::::::
/* fail-expr-vec1: sequence assignment, subselect, definition */
int main() {
  sequence a;
  a = [1000;];    /* sequence cannot be terminated by semi colon */
}
::::::::::::::
fail-type-float.mm
::::::::::::::
/* fail-type-float: float literals, operators and relationals */
int main() {
  float a;
  a = 1;     /* expect float contain decimal point */
}
```

6.2 Output of the Test Suite

```
$ ./testmm.sh && ./testall.sh
fail-constant-decl...OK
fail-constant-decl2...OK
fail-constant-decl3...OK
fail-expr-mat1...OK
fail-expr-mat2...OK
fail-expr-vec1...OK
fail-type-float...OK
test-constant-decl...OK
test-expr-mat...OK
test-expr-mat2...OK
test-expr-mat3...OK
test-expr-sizeof...OK
test-expr-vec...OK
test-func-mat...OK
test-globals...OK
test-mat-add...OK
test-mat-assign...OK
test-mat-augment...OK
test-mat-mul...OK
test-op-mat...OK
test-op-vec...OK
test-seq-assign...OK
test-seq-select...OK
test-type-cast...OK
test-type-float...OK
test-type-string...OK
test-var-init...OK
test-add1...OK
test-arith1...OK
test-arith2...OK
test-arith3...OK
test-fib...OK
test-for1...OK
```

```
test-for2...OK
test-func1...OK
test-func2...OK
test-func3...OK
test-func4...OK
test-func5...OK
test-func6...OK
test-func7...OK
test-func8...OK
test-gcd...OK
test-gcd2...OK
test-global1...OK
test-global2...OK
test-global3...OK
test-hello...OK
test-if1...OK
test-if2...OK
test-if3...OK
test-if4...OK
test-if5...OK
test-local1...OK
test-local2...OK
test-ops1...OK
test-ops2...OK
test-var1...OK
test-var2...OK
test-while1...OK
test-while2...OK
fail-assign1...OK
fail-assign2...OK
fail-assign3...OK
fail-dead1...OK
fail-dead2...OK
fail-expr1...OK
fail-expr2...OK
fail-for1...OK
fail-for2...OK
fail-for3...OK
fail-for4...OK
fail-for5...OK
fail-func1...OK
fail-func2...OK
fail-func3...OK
fail-func4...OK
fail-func5...OK
fail-func6...OK
fail-func7...OK
fail-func8...OK
fail-func9...OK
fail-global1...OK
fail-global2...OK
fail-if1...OK
fail-if2...OK
fail-if3...OK
fail-nomain...OK
fail-return1...OK
fail-return2...OK
fail-while1...OK
fail-while2...OK
```

7 References

- Alfred V. Aho, Monica Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, 2006. Second Edition.
- Brian W. Kernighan, and Dennis M. Ritchie. *The C Programming Language.* Prentice Hall, 1998. Second Edition.
- Stephen A. Edwards. *The MicroC Compiler.* Columbia University COMS4115 lecture slides.
- N. Devillard. *gnuplot interfaces in ANSI C.*

  Retrieved from `http://ndevilla.free.fr/gnuplot/`

8 Appendix

8.1 Code Listing for Compiler

MiniMat compiles to LLVM IR code, and comprises six OCaml program files (with non-blank lines, including comments, counted below):

```
$ grep -c "[^ ]" *ml*
ast.ml:131
codegen.ml:551
minimat.ml:19
parser.mly:149
scanner.mll:73
semant.ml:230
```

8.1.1 Abstract Syntax Tree

Listing 6: Compiler – ast.ml

```
(* Abstract Syntax Tree and functions for printing it *)
(* Minimat by Terence Lim tl2735@columbia.edu for COMS4115 *)

type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq |
          And | Or | Pow | Rem | Dotmul | Dotdiv | Dotrem | Dotpow

type uop = Neg | Not | Transpose

type typ = Int | Bool | Void | Float | Handle | String | Sequence | Matrix

type bind = typ * string

type decltyp = Declexternal | Declfunction | Declconstant

type expr =
    Literal of int
  | BoolLit of bool
  | FloatLit of float
  | StringLit of string
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of string * expr
  | Stride of expr * expr * expr
  | Seqselect of expr * expr
  | Seqassign of expr * expr * expr
  | Matselect of expr * expr * expr
  | Matassign of expr * expr * expr * expr
  | Call of string * expr list
  | SeqLit of expr list
  | MatLit of expr list list
  | Noexpr

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

type func_decl = {
    typ : typ;
    fname : string;
    formals : bind list;
    locals : bind list;
    body : stmt list;
    decltyp : decltyp;  (* local function | external function | constant *)
  }

type program = bind list * func_decl list

(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Pow -> "^"
  | Rem -> "%"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"
  | Dotmul -> ".*"
  | Dotdiv -> "./"
  | Dotrem -> ".%"
  | Dotpow -> ".^"

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

let rec string_of_expr = function
    Literal(l)      -> string_of_int l
  | FloatLit(f)     -> string_of_float f
  | StringLit(s)    -> "\"" ^ s ^ "\""
  | 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)        -> v ^ " = " ^ string_of_expr e
  | Stride(b, s, e) ->
      string_of_expr b ^ ":" ^ string_of_expr s ^ ":" ^ string_of_expr e
  | Seqselect(v, e)     -> string_of_expr v ^ "[" ^ string_of_expr e ^ "]"
  | Seqassign(v, e, x) ->
      string_of_expr v ^ "[" ^ string_of_expr e ^ "] = " ^ string_of_expr x
  | Matselect(v, e1, e2) -> string_of_expr v ^ "[" ^ string_of_expr e1
      ^ ", " ^ string_of_expr e2 ^ "]"
  | Matassign(v, e1, e2, x) -> string_of_expr v ^ "[" ^ string_of_expr e1
      ^ ", " ^ string_of_expr e2 ^ "] = " ^ string_of_expr x
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | SeqLit (el) -> "[" ^ String.concat ", " (List.map string_of_expr el) ^ "]"
  | MatLit (el) -> "[" ^ String.concat "; " (List.map (fun e2 ->
      String.concat ", " (List.map string_of_expr e2)) 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"
  | Handle   -> "handle"
  | Bool     -> "bool"
  | Void     -> "void"
  | Float    -> "float"
  | String   -> "string"
  | Sequence -> "sequence"
  | Matrix   -> "matrix"

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

let string_of_fdecl fdecl = match fdecl.decltyp with
| Declexternal ->  "external " ^ string_of_typ fdecl.typ ^ " " ^
    fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals) ^
    ");\n"
| Declfunction -> 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"
| Declconstant -> match (List.hd fdecl.body) with
    Return(e) -> "constant " ^ (string_of_typ fdecl.typ) ^ " " ^ fdecl.fname
```

```
      ^ " = " ^ (string_of_expr e) ^ ";\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)
```

8.1.2 OCamllex Scanner

Listing 7: Compiler – scanner.mll

```
(* Ocamllex scanner for MiniMat *)
(* Minimat by Terence Lim tl2735@columbia.edu for COMS4115 *)
{ open Parser
  let un_esc s =
        Scanf.sscanf ("\"" ^ s ^ "\"") "%S%!" (fun x -> x)
}

let letter = ['a'-'z' 'A'-'Z']
let digit  = ['0'-'9']
let esc    = '\\' ['\\' ''' '"' 'n' 'r' 't']
let esc_ch = ''' (esc) '''
let ascii  = ([' '-'!' '#'-'[' ']'-'~'])
let string = '"' ( (ascii | esc)* as s ) '"'

rule token = parse
  [' ' '\t' '\r' '\n']    { token lexbuf }   (* Whitespace *)
| "/*"                    { comment lexbuf } (* Comments *)
| "float"      { FLOAT }
| "string"     { STRING }
| "sequence"   { SEQUENCE }
| "matrix"     { MATRIX }
| "handle"     { HANDLE }
| "external"   { EXTERNAL }
| "constant"   { CONSTANT }
| "new"        { NEW }
|  ':'         { COLON }
|  "::"        { COCOLON }
|  '['         { LBRACK }
|  ']'         { RBRACK }
|  '\''        { TRANSPOSE }
|  ".*"        { DOTMUL }
|  "./"        { DOTDIV }
|  ".%"        { DOTREM }
|  ".^"        { DOTPOW }
|  '('         { LPAREN }
|  ')'         { RPAREN }
|  '{'         { LBRACE }
|  '}'         { RBRACE }
|  ';'         { SEMI }
|  ','         { COMMA }
|  '+'         { PLUS }
|  '-'         { MINUS }
| '*'          { TIMES }
| '/'          { DIVIDE }
| '^'          { POW }
| '%'          { REM }
| '='          { ASSIGN }
| "=="         { EQ }
| "!="         { NEQ }
| '<'          { LT }
| "<="         { LEQ }
| ">"          { GT }
| ">="         { GEQ }
| "&&"         { AND }
| "||"         { OR }
| "!"          { NOT }
| "if"         { IF }
| "else"       { ELSE }
| "for"        { FOR }
| "while"      { WHILE }
| "return"     { RETURN }
| "int"        { INT }
| "bool"       { BOOL }
| "void"       { VOID }
| "true"       { TRUE }
| "false"      { FALSE }
| digit+ as lxm                          { INTLIT(int_of_string lxm) }
| digit*['.']digit+ as lxm               { FLOATLIT(float_of_string lxm) }
| string                                 { STRINGLIT(un_esc s) }
| letter (letter | digit | '_')* as lxm { ID(lxm) }
| eof                                    { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

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

8.1.3 OCamlyacc Parser

Listing 8: Compiler – parser.mly

```
/* Ocamlyacc parser for MiniMat */
/* Minimat by Terence Lim tl2735@columbia.edu for COMS4115 */

%{
open Ast
(* Uncomment next line to trace all states parsed:
let _ = Parsing.set_trace true
*)
%}

%token SEQUENCE MATRIX COLON COCOLON LBRACK RBRACK
%token FLOAT TRANSPOSE STRING
%token EXTERNAL HANDLE CONSTANT NEW
%token POW REM DOTDIV DOTMUL DOTREM DOTPOW
```

```
%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA
%token PLUS MINUS TIMES DIVIDE ASSIGN NOT
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
%token RETURN IF ELSE FOR WHILE INT BOOL VOID
%token <int> INTLIT
%token <string> ID
%token <float> FLOATLIT
%token <string> STRINGLIT
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left COLON COCOLON
%left PLUS MINUS
%left TIMES DIVIDE REM DOTMUL DOTDIV DOTREM
%left POW DOTPOW
%right NOT NEG
%left TRANSPOSE

%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }

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

fdecl:
   typ ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE
     { { typ = $1;
         fname = $2;
         formals = $4;
         locals = List.rev $7;
         body = List.rev $8;
         decltyp = Declfunction } }
|  EXTERNAL typ ID LPAREN formals_opt RPAREN SEMI
     { { typ = $2;
         fname = $3;
         formals = $5;
         locals = [];
         body = [];
         decltyp = Declexternal } } /* external function declaration */
|  CONSTANT typ ID ASSIGN expr SEMI
     { { typ = $2;
```

```
        fname =  "%"^$3;
        formals = [];
        locals = [];
        body =  [Return($5)];
        decltyp = Declconstant } } /* global constant, prefix with "%" */

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 }
  | BOOL     { Bool }
  | HANDLE   { Handle }
  | FLOAT    { Float }
  | STRING   { String }
  | SEQUENCE { Sequence }
  | MATRIX   { Matrix }
  | VOID     { Void }

vdecl_list:
    /* nothing */    { [] }
  | vdecl_list vdecl { $2 :: $1 }

vdecl:
   typ ID SEMI { ($1, $2) }

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([])) }
  | 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) }

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

expr:
    INTLIT           { Literal($1) }
  | FLOATLIT         { FloatLit($1) }
  | STRINGLIT        { StringLit($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 POW    expr { Binop($1, Pow,   $3) }
  | expr REM    expr { Binop($1, Rem,   $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 DOTMUL expr { Binop($1, Dotmul,  $3) }
  | expr DOTDIV expr { Binop($1, Dotdiv,   $3) }
  | expr DOTREM expr { Binop($1, Dotrem,   $3) }
  | expr DOTPOW expr { Binop($1, Dotpow,   $3) }
  | expr TRANSPOSE   { Unop(Transpose,$1) }
  | LBRACK rows SEMI RBRACK                      { MatLit(List.rev $2) }
  | LBRACK actuals_opt RBRACK                    { SeqLit($2) }
  | ID LBRACK expr COMMA expr RBRACK ASSIGN expr { Matassign(Id($1),$3,$5,$8)}
  | ID LBRACK expr COMMA expr RBRACK             { Matselect(Id($1),$3,$5) }
  | ID LBRACK expr RBRACK ASSIGN expr            { Seqassign(Id($1),$3,$6) }
  | ID LBRACK expr RBRACK                        { Seqselect(Id($1),$3) }
  | expr COLON expr COLON expr                   { Stride($1,$3,$5) }
  | expr COCOLON expr                            { Stride($1,Literal(1),$3) }
  | MINUS expr %prec NEG                         { Unop(Neg, $2) }
  | NOT expr                                     { Unop(Not, $2) }
  | ID ASSIGN expr                               { Assign($1, $3) }
  | NEW typ LPAREN actuals_opt RPAREN            { Call(string_of_typ $2, $4)}
  | ID LPAREN actuals_opt RPAREN                 { Call($1, $3) }
  | LPAREN expr RPAREN                           { $2 }

actuals_opt:
    /* nothing */ { [] }
  | actuals_list  { List.rev $1 }

actuals_list:
    expr                    { [$1] }
  | actuals_list COMMA expr { $3 :: $1 }

rows:
    actuals_opt            { [$1] }
  | rows SEMI actuals_opt  { $3 :: $1 }
```

8.1.4 Semantic Analysis

Listing 9: Compiler – semant.ml

```
(* Semantic checking for the MiniMat compiler *)
(* Minimat by Terence Lim tl2735@columbia.edu for COMS4115 *)

open Ast

module StringMap = Map.Make(String)

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

   Check each global variable, then check each function *)

let check (globals, 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
  (* Raise an exception of the given rvalue type cannot be assigned to
     the given lvalue type *)
  let check_assign lvaluet rvaluet err =
     if lvaluet == rvaluet then lvaluet else raise err
  in
  (*------------------------------------------------------------------
     Raise an exception of the given expression is not of the given type
  -------------------------------------------------------------------*)
  let check_type e t typ_list =
    if not (List.mem t typ_list) then raise (Failure ("illegal type "
                           ^ string_of_typ t ^ " of " ^ string_of_expr e))
  in
  (**** Checking Global Variables ****)
  List.iter (check_not_void (fun n -> "illegal void global " ^ n)) globals;

  report_duplicate (fun n -> "duplicate global " ^ n) (List.map snd globals);

  (**** Checking Global Constants ****)
  report_duplicate (fun n -> "duplicate constant " ^ n)
    (List.map (fun fd -> String.sub fd.fname 1 (String.length fd.fname - 1))
       (List.filter (fun s -> s.decltyp = Declconstant) functions));

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

  if List.mem "print" (List.map (fun fd -> fd.fname) functions)
  then raise (Failure ("function print may not be defined")) else ();

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

  (* Function declaration for a named function *)

  let built_in_decls =
    List.fold_left (fun m (fdname, fdtyp, fdforms) -> StringMap.add fdname
                    {typ = fdtyp; fname = fdname; formals = fdforms;
                     locals = []; body = []; decltyp = Declfunction}
                    m)
                  StringMap.empty
    [("print", Void, [(Int, "x")]);
     ("float_of_int", Float, [(Int, "x")]);
     ("int_of_float", Int, [(Float, "x")]);
     ("int_of_seq", Int, [(Sequence, "x")]);
     ("float_of_mat", Float, [(Matrix, "x")]);
     ("cols", Int, [(Matrix, "x")]);
     ("matrix", Matrix, [(Int, "x"); (Int, "y")]);
     ("sequence", Sequence, [(Int, "y")]);
     ("printb", Void, [(Bool, "x")]);]
  in
  let function_decls = List.fold_left (fun m fd ->
    StringMap.add fd.fname fd m) built_in_decls functions
  in
  let function_decl s = try StringMap.find s function_decls
       with Not_found -> raise (Failure ("unrecognized function " ^ s))
  in
  let _ = function_decl "main" in (* Ensure "main" is defined *)

  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);

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

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

    (* Type of each variable (global, formal, or local *)
    let symbols = List.fold_left (fun m (t, n) -> StringMap.add n t m)
        StringMap.empty (globals @ func.formals @ func.locals )
    in
    let type_of_identifier s =
      try StringMap.find s symbols
      with Not_found -> raise (Failure ("undeclared identifier " ^ s))
    in

    (* Global constants are function decls with decltyp Declconstant *)
    let type_of_constant s =
      let fd = function_decl s in match fd.decltyp with Declconstant -> fd.typ
      | _ -> raise (Failure ("undeclared identifier " ^ s))
    in

    (* Return the type of an expression or throw an exception *)
    let rec expr = function
        Literal _   -> Int
      | BoolLit _   -> Bool
      | FloatLit _  -> Float
      | StringLit _ -> String
      | Id s -> if (func.decltyp = Declconstant) then
          (* global constant definitions cannot comprise other identifiers *)
          raise (Failure ("constant " ^ (String.sub func.fname 1
            (String.length func.fname - 1)) ^
            " cannot be defined with an identifier " ^ s));
          if (StringMap.mem s symbols) then type_of_identifier s
          (* Global constant identifiers are internally prefixed by "%" *)
          else if (StringMap.mem ("%"^s) function_decls)
          then type_of_constant ("%"^s)
          else raise (Failure ("undeclared identifier " ^ s))

      (*------------------------------------------------------------------
         Checks inside matrix, which is a list of list of floats or matrices
      ------------------------------------------------------------------*)
      | MatLit(e) -> List.iter (fun e2 -> List.iter (fun e1 ->
          check_type e1 (expr e1) [Float; Matrix]) e2) e;
          Matrix

      (*------------------------------------------------------------------
         Checks inside sequence, which is a list of ints or sequences
      ------------------------------------------------------------------*)
      | SeqLit(e) ->
          List.iter (fun i -> check_type i (expr i) [Int; Sequence]) e;
          Sequence

      (*------------------------------------------------------------------
         checks sequence colon expression b:s:e, which must be ints
      ------------------------------------------------------------------*)
      | Stride(b, s, e) as ex -> check_type ex (expr b) [Int];
          check_type ex (expr s) [Int];
          check_type ex (expr e) [Int];
          Sequence

      | Binop(e1, op, e2) as e -> let t1 = expr e1 and t2 = expr e2 in
        (match op with
          Add | Sub | Mult | Div | Rem when (t1 = t2) &&
            (t1 = Int || t1 = Float || t1 = Sequence || t1 = Matrix) -> t1
        | Pow when (t1 = Float && t2 = Float) ||
            (t1 = Matrix && t2 = Int) -> t1
        | Dotmul | Dotdiv | Dotrem | Dotpow
            when (t1 = t2) && t1 = Matrix -> t1
        | Less | Leq | Greater | Geq | Equal | Neq when t1 = t2 &&
            t1 != Sequence -> if t1 = Matrix then Sequence else Bool
        | And | Or when (t1 = t2) && t1 = Bool -> t1
        | _ -> raise (Failure ("illegal binary operator " ^
              string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
              string_of_typ t2 ^ " in " ^ string_of_expr e))
        )
      | Unop(op, e) as ex -> let t = expr e in
         (match op with
         | Neg when (t = Int || t = Sequence || t = Float || t = Matrix) -> t
         | Not when t = Bool || t = Sequence -> Bool
         | Transpose when t = Matrix -> t    (* MATRIX *)
         | _ -> raise (Failure ("illegal unary operator " ^ string_of_uop op
                        ^ string_of_typ t ^ " in " ^ string_of_expr ex)))
      | Noexpr -> Void
      | Assign(var, e) as ex -> let lt = type_of_identifier var
                                 and rt = expr e in
        check_assign lt rt (Failure ("illegal assignment " ^ string_of_typ lt
                                     ^ " = " ^ string_of_typ rt ^ " in "
                                     ^ string_of_expr ex))

      (*-----------------------------------------------------
         Check matrix assignment by [row,column] statement: A[i,j] = rhs
         Allow rhs to be a float or matrix.
         Col and row indexes i,j can be int or sequence
      -----------------------------------------------------*)
      | Matassign(var, e1, e2, v) as ex -> check_type ex (expr var) [Matrix];
          check_type ex (expr e1) [Int; Sequence];
          check_type ex (expr e2) [Int; Sequence];
          check_type ex (expr v)  [Matrix];
          Matrix

      (*-----------------------------------------------------
         Check matrix subselect statement: A[i,j]
         Col and row indexes i,j can be int or sequence
      -----------------------------------------------------*)
      | Matselect(var, e1, e2) as ex -> check_type ex (expr var) [Matrix];
          check_type ex (expr e1) [Int; Sequence];
          check_type ex (expr e2) [Int; Sequence];
          Matrix

      (*-----------------------------------------------------
         Check sequence assignment statement: A[i] = rhs
         Index i can be int or sequence
       -----------------------------------------------------*)
      | Seqassign(var, e, v) as ex ->
          check_type ex (expr var) [Sequence];
          check_type ex (expr e) [Int; Sequence];
          check_type ex (expr v) [Sequence];
          Sequence

      (*-----------------------------------------------------
        Checks sequence subselect statement: V[i]
        Index i can be int or sequence
     ---------------------------------------------------------*)
      | Seqselect(var, e) as ex ->
        check_type ex (expr var) [Sequence];
        check_type ex (expr e) [Int; Sequence];
        Sequence

      | Call("printf", _)            -> Void
      | Call("string", _)            -> String
      | Call("length", [e]) as ex    ->
        ignore(check_type ex (expr e) [Sequence; Matrix]); Int
      | Call(fname, actuals) as call -> let fd = function_decl fname in
        if List.length actuals != List.length fd.formals then
          raise (Failure ("expecting " ^ string_of_int (List.length
             fd.formals) ^ " arguments in " ^ string_of_expr call))
        else
          List.iter2 (fun (ft, _) e -> let et = expr e in
          ignore (check_assign ft et
                    (Failure ("illegal actual argument found " ^
                              string_of_typ et ^ " expected " ^
                              string_of_typ ft ^ " in " ^ string_of_expr e))))
            fd.formals actuals;
          fd.typ
    in

    let check_bool_expr e = if expr e != Bool then
      raise (Failure ("expected Boolean expression in " ^ string_of_expr e))
    else () in

    (* Verify a statement or throw an exception *)
    let rec stmt = function
        Block sl -> let rec check_block = function
            [Return _ as s] -> stmt s
          | Return _ :: _ -> raise (Failure "nothing may follow a return")
          | Block sl :: ss -> check_block (sl @ ss)
          | s :: ss -> stmt s ; check_block ss
          | [] -> ()
        in check_block sl
      | Expr e -> ignore (expr e)
      | Return e -> let t = expr e in if t = func.typ then () else
        raise (Failure ("return gives " ^ string_of_typ t ^ " expected " ^
                        string_of_typ func.typ ^ " in " ^ string_of_expr e))

      | If(p, b1, b2) -> check_bool_expr p; stmt b1; stmt b2
      | For(e1, e2, e3, st) -> ignore (expr e1); check_bool_expr e2;
        ignore (expr e3); stmt st
      | While(p, s) -> check_bool_expr p; stmt s
    in
    stmt (Block func.body)
  in
  List.iter check_function functions
```

8.1.5 Code Generation

Listing 10: Compiler – codegen.ml

```
(* Code generation: takes a semantically checked AST and produces LLVM IR *)
(* Minimat by Terence Lim tl2735@columbia.edu for COMS4115 *)

module L = Llvm
module A = Ast

module StringMap = Map.Make(String)

let translate (globals, functions) =
  let context      = L.global_context () in
  let the_module   = L.create_module context "MiniMat"
  and double_t     = L.double_type context
  and i64_t        = L.i64_type  context
  and i32_t        = L.i32_type  context
  and i8_t         = L.i8_type   context
  and i1_t         = L.i1_type   context
  and void_t       = L.void_type context in

  let i8ptr_t      = L.pointer_type i8_t
  and i32ptr_t     = L.pointer_type i32_t
  and string_t     = L.pointer_type i8_t
  and sequence_t   = L.pointer_type i32_t
  and matrix_t     = L.pointer_type double_t
  and handle_t     = L.pointer_type i64_t in

  let ltype_of_typ = function
      A.Int        -> i32_t
    | A.Handle     -> handle_t
    | A.Bool       -> i1_t
    | A.Float      -> double_t
    | A.Sequence   -> sequence_t
    | A.Matrix     -> matrix_t
    | A.String     -> string_t
    | A.Void       -> void_t in

(* ----------------------------------------------------------------------
  Declare external functions required in codegen
------------------------------------------------------------------------*)
  let printf_t = L.var_arg_function_type i32_t [| i8ptr_t |] in
  let printf_func = L.declare_function "printf" printf_t the_module in

  let snprintf_t = L.var_arg_function_type i32_t [| i8ptr_t |] in
  let snprintf = L.declare_function "snprintf" snprintf_t the_module in

  let memset_t = L.function_type void_t [| i8ptr_t; i32_t; i32_t|] in
  let memset = L.declare_function "memset" memset_t the_module in

  let memcpy_t = L.function_type i32_t [| i8ptr_t; i8ptr_t; i32_t|] in
  let memcpy = L.declare_function "memcpy" memcpy_t the_module in

```

```
(*------------------------------------------------------------------------
  Instructions to get/set size info of matrix or sequence object
------------------------------------------------------------------------*)
  let sizeof_offset = L.const_int i32_t (-1)  (* # bytes for storage *)
  and length_offset = L.const_int i32_t (-2)  (* length of sequence *)
  and cols_offset   = L.const_int i32_t (-3)  (* # columns of matrix *)
  and rows_offset   = L.const_int i32_t (-4)  (* # rows of matrix *)
  and string_sz     = L.const_int i32_t 256   (* max chars in string *)
  and int_sz        = L.const_int i32_t 4
  and double_sz     = L.const_int i32_t 8
  and one_32t       = L.const_int i32_t 1
  and zero_32t      = L.const_int i32_t 0 in

  let getdim from_ptr item the_builder =
    let loc = L.build_bitcast from_ptr i32ptr_t "dim" the_builder in
    let loc = L.build_gep loc [| item |] "dim" the_builder in
    L.build_load loc "dim" the_builder in

  let putdim from_ptr item the_val the_builder =
    let loc = L.build_bitcast from_ptr i32ptr_t "dim" the_builder in
    let loc = L.build_gep loc [| item |]  "dim" the_builder in
    L.build_store the_val loc the_builder in

(*------------------------------------------------------------------------
  Instructions to allocate storage for new matrix, sequence or string.
  Prepend matrix and sequence object with 16-byte header containing size info.
------------------------------------------------------------------------*)
  let head_from_body loc the_builder =
    let charptr = L.build_bitcast loc i8ptr_t "new" the_builder in
    L.build_gep charptr [| (L.const_int i8_t (-16)) |]  "new" the_builder in

  let body_from_head loc the_builder =
    let charptr = L.build_bitcast loc i8ptr_t "new" the_builder in
    L.build_gep charptr [| (L.const_int i8_t (16)) |]  "new" the_builder in

  let select_heap = true and select_stack = false in (* pick heap or stack *)

  (* allocate a block of sz bytes from stack or heap *)
  let build_new stack_or_heap sz the_builder =
    let ch_ptr = (if stack_or_heap = select_heap then
      L.build_array_malloc i8_t sz "new" the_builder else
      L.build_array_alloca i8_t sz "new" the_builder) in
    ignore (L.build_call memset [| ch_ptr ; zero_32t ; sz |] "" the_builder);
    ch_ptr in

  (* allocate sz elements, each of len bytes from stack or heap *)
  let build_vecnew stack_or_heap len sz the_builder =
    let sz = L.build_mul len sz "new" the_builder in
    let alloc_sz = L.build_add sz (L.const_int i32_t 16) "new" the_builder in
    let char_ptr = build_new stack_or_heap alloc_sz the_builder in
    let vec_ptr = body_from_head char_ptr the_builder in
    ignore (putdim vec_ptr sizeof_offset alloc_sz the_builder);
    ignore (putdim vec_ptr length_offset len the_builder);
    ignore (putdim vec_ptr rows_offset zero_32t the_builder);
    ignore (putdim vec_ptr cols_offset zero_32t the_builder);
    L.build_bitcast vec_ptr sequence_t "new" the_builder in

  (* allocate row * col elements of double_sz bytes from stack or heap *)
  let build_matnew stack_or_heap row col the_builder =
    let len = L.build_mul row col "new" the_builder in
    let vec_ptr = build_vecnew stack_or_heap len double_sz the_builder in
    ignore (putdim vec_ptr rows_offset row the_builder);
    ignore (putdim vec_ptr cols_offset col the_builder);
    L.build_bitcast vec_ptr matrix_t "new" the_builder in

(*-----------------------------------------------------------------
   To put or get a data item from matrix or sequence
-------------------------------------------------------------------*)
  let build_put from_ptr offset the_val the_builder =
    let loc = L.build_gep from_ptr [| offset |] "put" the_builder in
    L.build_store the_val loc the_builder in

  let build_get from_ptr offset the_builder =
    let loc = L.build_gep from_ptr [| offset |] "get" the_builder in
    L.build_load loc "get" the_builder in

  let build_getrc from_ptr row col the_builder =
    let offset = getdim from_ptr cols_offset the_builder in
    let offset = L.build_mul row offset "get" the_builder in
    let offset = L.build_add col offset "get" the_builder in
    build_get from_ptr offset the_builder in

  let build_putrc from_ptr row col the_val the_builder =
    let offset = getdim from_ptr cols_offset the_builder in
    let offset = L.build_mul row offset "getrc" the_builder in
    let offset = L.build_add col offset "getrc" the_builder in
    build_put from_ptr offset the_val the_builder in

  let build_seq_of_int the_val the_builder =
    let to_ptr = build_vecnew select_stack one_32t int_sz the_builder
    in ignore(build_put to_ptr zero_32t the_val the_builder);
    to_ptr in

  let build_mat_of_float the_val the_builder =
    let to_ptr = build_matnew select_stack one_32t one_32t the_builder
    in ignore(build_put to_ptr zero_32t the_val the_builder);
    to_ptr in

(*----------------------------------------------------------------------
   Declare each global variable; remember its value in a map
------------------------------------------------------------------------*)
  let null_t = L.define_global "_null" (L.const_stringz context "") the_module
  in let build_const_init = function
      | A.Float     -> L.const_float double_t 0.0
      | A.Sequence  -> L.const_null sequence_t
      | A.Matrix    -> L.const_null matrix_t
      | A.String    -> L.const_bitcast null_t string_t
      | A.Handle    -> L.const_null handle_t
    | _ as t       -> L.const_int (ltype_of_typ t) 0
  in

  let global_vars =
    let global_var m (t, n) =
      let init = build_const_init t
      in StringMap.add n (L.define_global n init the_module) m in
    List.fold_left global_var StringMap.empty globals in

(*-------------------------------------------------------------------
   Populate lists of local functions, externals declarations, and constants
-------------------------------------------------------------------*)
  let local_functions =
     List.filter (fun fdecl -> fdecl.A.decltyp = A.Declfunction) functions
  and external_functions =
     List.filter (fun fdecl -> fdecl.A.decltyp = A.Declexternal) functions
  and constant_functions =
     List.filter (fun fdecl -> fdecl.A.decltyp = A.Declconstant) functions in

  let constant_decls =
     let constant_decl m fdecl =
      let name = fdecl.A.fname and
        e1 = (match (List.hd fdecl.A.body) with
          A.Return(e) -> e | _ -> A.Noexpr) in
      StringMap.add name e1 m in
     List.fold_left constant_decl StringMap.empty constant_functions in

  let external_decls =
    let external_decl m fdecl =
      let name = fdecl.A.fname and
          formal_types = Array.of_list (List.map (fun (t,_) -> ltype_of_typ t)
                                                  fdecl.A.formals)
      in
      let ftype = L.function_type (ltype_of_typ fdecl.A.typ) formal_types
      in
      StringMap.add name (L.declare_function name ftype the_module, fdecl) m
    in
    List.fold_left external_decl StringMap.empty external_functions in

  let function_decls =
    let function_decl m fdecl =
      let name = fdecl.A.fname
      and formal_types = Array.of_list
          (List.map (fun (t,_) -> ltype_of_typ t) fdecl.A.formals) in
    let ftype = L.function_type (ltype_of_typ fdecl.A.typ) formal_types in
    StringMap.add name (L.define_function name ftype the_module, fdecl) m in
    List.fold_left function_decl StringMap.empty local_functions in

(*-------------------------------------------------------------------
  Helper instructions to call local or external functions by name
-------------------------------------------------------------------*)
    let select_free = true and select_keep = false in (* to free mem or not *)

    (* if mat/seq/string, copy values to heap or stack, else return value *)
    let build_copy ans heap_or_stack free_or_keep the_builder =
      let t = L.type_of ans in
      (if (t = matrix_t || t = sequence_t) then
        let siz = getdim ans sizeof_offset the_builder in
        let dst = build_new heap_or_stack siz the_builder
        and src = head_from_body ans the_builder in
        ignore(L.build_call memcpy [| dst; src; siz |] "" the_builder);
        ignore( if free_or_keep = select_free then
          L.build_free src the_builder else src);
        L.build_bitcast (body_from_head dst the_builder) t "cp" the_builder
      else if (t = string_t) then
        let dst = build_new heap_or_stack string_sz the_builder in
        ignore(L.build_call memcpy [| dst; ans; string_sz |] "" the_builder);
        ignore( if free_or_keep = select_free then
          L.build_free ans the_builder else ans);
        dst
      else ans) in

    (* call a locally-defined function by name *)
    let build_funcall f actuals_array the_builder=
      let (fdef, fdecl) = (try StringMap.find f function_decls with
        Not_found -> raise (Failure("Not Found " ^ f))) in
      let result = (match fdecl.A.typ with A.Void -> "" | _ -> f ^ "_res") in
      let ans = L.build_call fdef actuals_array result the_builder
      (* callee returned mat/seq/string in heap, so copy to stack and free *)
      in build_copy ans select_stack select_free the_builder in

    (* call an externally-declared function by name *)
    let build_external fname actuals_array the_builder=
      let (fdef, fdecl) = (try StringMap.find fname external_decls with
        Not_found -> raise(Failure("Not Found" ^ fname))) in
      let result = (match fdecl.A.typ with A.Void -> "" | _ -> fname ^ "_res")
      in L.build_call fdef actuals_array result the_builder in

(*---------------------------------------------------------------------------
  Main "inner loop" to iterate on each function record
---------------------------------------------------------------------------*)
  (* Fill in the body of the given function *)
  let build_function_body fdecl =
    let (the_function, _) = StringMap.find fdecl.A.fname function_decls in
    let builder = L.builder_at_end context (L.entry_block the_function) in

    (* 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;
        let local = L.build_alloca (ltype_of_typ t) n builder in
        ignore (L.build_store p local builder);
        StringMap.add n local m in

      let add_local m (t, n) =
        let local_var = L.build_alloca (ltype_of_typ t) n builder in
        ignore (L.build_store (build_const_init t) local_var builder);
        StringMap.add n local_var m in

      let formals = List.fold_left2 add_formal StringMap.empty fdecl.A.formals
          (Array.to_list (L.params the_function)) in
      List.fold_left add_local formals fdecl.A.locals in

    (* Return the value for a variable or formal argument *)
    let lookup n = try StringMap.find n local_vars
                   with Not_found -> StringMap.find n global_vars
    in

    (*---------------------------------------------------------------------
      Construct code for an expression; return its value
    ---------------------------------------------------------------------*)
    let rec expr builder = function
        A.Literal i   -> L.const_int i32_t i
      | A.FloatLit f  -> L.const_float double_t f
      | A.BoolLit b   -> L.const_int i1_t (if b then 1 else 0)
      | A.StringLit s -> L.build_global_stringptr s "str" builder
      | A.Noexpr      -> L.const_int i32_t 0
      | A.Id s        ->
          (* Id may be in local vars, global vars, or constants lists *)
          if (StringMap.mem s local_vars || StringMap.mem s global_vars)
          then L.build_load (lookup s) s builder
          else expr builder (try StringMap.find ("%"^s) constant_decls
          with Not_found -> raise(Failure("ID Not Found " ^ s)))

      (*--------------------------------------------------------------------
        To construct matrix literal, by folding over rows and columns.
        Calls vertcat() and horzcat() helper functions in standard library
      -------------------------------------------------------------------*)
      | A.MatLit (act) ->
          let v0 = build_matnew select_stack zero_32t zero_32t builder in
          let catadj leftmat right =
            let rightmat = (if (L.type_of right) != matrix_t then
              build_mat_of_float right builder else right) in
            build_funcall "horzcat" [| leftmat; rightmat |] builder in
          let makerow row =
            let actuals = List.rev (List.map (expr builder) (List.rev row)) in
            List.fold_left catadj v0 actuals in
          let rows = List.rev (List.map makerow (List.rev act)) in
          List.fold_left (fun toprow botrow ->
            build_funcall "vertcat" [| toprow; botrow |] builder)
            v0 rows

      (*--------------------------------------------------------------------
        Construct sequence literal, by calling append() to fold over list
      -------------------------------------------------------------------*)
      | A.SeqLit (act) ->
          let v0 = build_vecnew select_stack zero_32t int_sz builder and
              actuals = List.rev (List.map (expr builder) (List.rev act)) in
          List.fold_left (fun v1 v2 ->
            let v3 = (if (L.type_of v2) != sequence_t
            then build_seq_of_int v2 builder else v2)
            in build_funcall "append" [| v1; v3 |] builder)
            v0 actuals

      (*-------------------------------------------------------------------
        Construct sequence colon expression, by calling stride() helper func
      -------------------------------------------------------------------*)
      | A.Stride (b, s, e) -> let b1 = expr builder b and
            s1 = expr builder s and e1 = expr builder e in
        build_funcall "stride" [| b1; s1; e1 |] builder

      | A.Binop (e1, op, e2) ->
          let e3 = expr builder e1
          and e4 = expr builder e2 in
          let typ = L.type_of e3 in

          (* operands of sequence type *)
          (if typ = sequence_t then (match op with
            A.Add     -> build_funcall "vadd" [| e3; e4 |] builder
          | A.Sub     -> build_funcall "vsub" [| e3; e4 |] builder
          | A.Mult    -> build_funcall "vmul" [| e3; e4 |] builder
          | A.Div     -> build_funcall "vdiv" [| e3; e4 |] builder
          | A.Rem     -> build_funcall "vrem" [| e3; e4 |] builder
          | _ -> raise (Failure ((A.string_of_op op) ^ " not defined for "
                               ^ (L.string_of_lltype typ) ^ " in "
                               ^ (A.string_of_expr e2)))
              )
          (* operands of matrix type *)
          else if typ = matrix_t then (match op with
            A.Add     -> build_funcall "madd"    [| e3; e4 |] builder
          | A.Sub     -> build_funcall "msub"    [| e3; e4 |] builder
          | A.Mult    -> build_funcall "mmul"    [| e3; e4 |] builder
          | A.Div     -> build_funcall "mdiv"    [| e3; e4 |] builder
          | A.Rem     -> build_funcall "mrem"    [| e3; e4 |] builder
          | A.Pow     -> build_funcall "mpow"    [| e3; e4 |] builder
          | A.Equal   -> build_funcall "meq"     [| e3; e4 |] builder
          | A.Neq     -> build_funcall "mne"     [| e3; e4 |] builder
          | A.Less    -> build_funcall "mlt"     [| e3; e4 |] builder
          | A.Leq     -> build_funcall "mle"     [| e3; e4 |] builder
          | A.Greater -> build_funcall "mgt"     [| e3; e4 |] builder
          | A.Geq     -> build_funcall "mge"     [| e3; e4 |] builder
          | A.Dotmul  -> build_funcall "mdotmul" [| e3; e4 |] builder
          | A.Dotdiv  -> build_funcall "mdotdiv" [| e3; e4 |] builder
          | A.Dotrem  -> build_funcall "mdotrem" [| e3; e4 |] builder
          | A.Dotpow  -> build_funcall "mdotpow" [| e3; e4 |] builder
          | _ -> raise (Failure ((A.string_of_op op) ^ " not defined for "
                               ^ (L.string_of_lltype typ) ^ " in "
                               ^ (A.string_of_expr e2)))
              )
          (* operands of float type *)
          else if typ = double_t then (match op with
            A.Add     -> L.build_fadd e3 e4 "tmp" builder
          | A.Sub     -> L.build_fsub e3 e4 "tmp" builder
          | A.Mult    -> L.build_fmul e3 e4 "tmp" builder
          | A.Div     -> L.build_fdiv e3 e4 "tmp" builder
```

```
          | A.Rem     -> L.build_frem e3 e4 "tmp" builder
          | A.Pow     -> build_external "pow" [| e3; e4 |] builder
          | A.Equal   -> L.build_fcmp L.Fcmp.Ueq e3 e4 "tmp" builder
          | A.Neq     -> L.build_fcmp L.Fcmp.Une e3 e4 "tmp" builder
          | A.Less    -> L.build_fcmp L.Fcmp.Ult e3 e4 "tmp" builder
          | A.Leq     -> L.build_fcmp L.Fcmp.Ule e3 e4 "tmp" builder
          | A.Greater -> L.build_fcmp L.Fcmp.Ugt e3 e4 "tmp" builder
          | A.Geq     -> L.build_fcmp L.Fcmp.Uge e3 e4 "tmp" builder
          | _ -> raise (Failure ((A.string_of_op op) ^ " not defined for "
                               ^ (L.string_of_lltype typ) ^ " in "
                               ^ (A.string_of_expr e2)))
                 )
          (* operands of string type *)
          else if typ = string_t then (match op with
          | A.Equal   -> build_funcall "stringeq" [| e3; e4 |] builder
          | A.Neq     -> build_funcall "stringne" [| e3; e4 |] builder
          | A.Less    -> build_funcall "stringlt" [| e3; e4 |] builder
          | A.Leq     -> build_funcall "stringle" [| e3; e4 |] builder
          | A.Greater -> build_funcall "stringgt" [| e3; e4 |] builder
          | A.Geq     -> build_funcall "stringge" [| e3; e4 |] builder
          | _ -> raise (Failure ((A.string_of_op op) ^ " not defined for "
                               ^ (L.string_of_lltype typ) ^ " in "
                               ^ (A.string_of_expr e2)))
                 )
          else (match op with
            A.Add     -> L.build_add
          | A.Sub     -> L.build_sub
          | A.Mult    -> L.build_mul
          | A.Div     -> L.build_sdiv
          | A.Rem     -> L.build_srem
          | 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
          | A.Less    -> L.build_icmp L.Icmp.Slt
          | A.Leq     -> L.build_icmp L.Icmp.Sle
          | A.Greater -> L.build_icmp L.Icmp.Sgt
          | A.Geq     -> L.build_icmp L.Icmp.Sge
          | _ -> raise (Failure ((A.string_of_op op) ^ " not defined for "
                               ^ (L.string_of_lltype typ) ^ " in "
                               ^ (A.string_of_expr e2)))
                 ) e3 e4 "tmp" builder
               )
      | A.Unop(op, e) ->
          let e' = expr builder e in
          (match op with
          | A.Neg       -> let t = L.type_of e' in
            if (t = double_t) then L.build_fneg e' "tmp" builder
            else if (t = sequence_t) then build_funcall "vneg" [|e'|] builder
            else if (t = matrix_t) then build_funcall "mneg" [|e'|] builder
            else L.build_neg e'"tmp" builder
          | A.Transpose -> build_funcall "mtransp" [| e' |] builder
          | A.Not       ->
              (if (L.type_of e') = i1_t then L.build_not e' "tmp" builder
            else build_funcall "vnot" [| e' |] builder))

      (*-------------------------------------------------------------------
         When assigning from matrix/sequence/string r-value, copy its values
         (to stack, but to heap when assigning to global variable identifier)
      ---------------------------------------------------------------------*)
      | A.Assign (lv, rv) -> let rv1 = expr builder rv in
        let rv2 = (match rv with
          (* when r-value is identifier of mat/seq/str, then copy values *)
          A.Id(_) -> build_copy rv1 select_stack select_keep builder
        | _ -> rv1) in let rv3 = (if (StringMap.mem lv local_vars) then rv2
            (* if l-value is global id, then make a copy to heap *)
        else build_copy rv2 select_heap select_keep builder)
        in ignore (L.build_store rv3 (lookup lv) builder); rv3

      (*-------------------------------------------------------------------
         Subselect from matrix or sequence with multiple index positions.
         Requires mselect() and vselect() helper functions
       ---------------------------------------------------------------------*)
      | A.Matselect (s, r, c) -> let r1 = expr builder r and
            c1 = expr builder c and s1 = expr builder s in
        if (L.type_of r1 = i32_t && L.type_of c1 = i32_t)
        then (ignore(build_funcall "checkmatrc" [| s1; r1; c1|] builder);
              let v1 = build_getrc s1 r1 c1 builder in
              build_mat_of_float v1 builder)
        else let r2 = (if (L.type_of r1) != sequence_t then
          build_seq_of_int r1 builder else r1) and
            c2 = (if (L.type_of c1) != sequence_t then
              build_seq_of_int c1 builder else c1)
        in build_funcall "mselect" [| s1; r2; c2 |] builder;

      | A.Seqselect (s, e) ->
          let e1 = expr builder e and s1 = expr builder s in
          if (L.type_of e1) = i32_t
          then (ignore(build_funcall "checkseqlength" [| s1; e1 |] builder);
                let v1 = build_get s1 e1 builder in
                build_seq_of_int v1 builder)
          else build_funcall "vselect" [| s1; e1 |] builder

      (*-------------------------------------------------------------------
         Assign to multiple positions in a matrix or sequence
         Requires massign() and vassign() helper functions in standard lib
      ---------------------------------------------------------------------*)
      | A.Matassign (s, r, c, v) -> let r1 = expr builder r and c1 =
          expr builder c and s1 = expr builder s and v1 = expr builder v in
        if (L.type_of r1 = i32_t && L.type_of c1 = i32_t)
        (* directly put when index r and c are ints *)
        then (ignore(build_funcall "checkmatrc" [| s1; r1; c1|] builder);
              ignore(build_funcall "checkmatscalar" [| v1 |] builder);
              let v2 = build_get v1 zero_32t builder
              in ignore(build_putrc s1 r1 c1 v2 builder); v1)
        else let r2 = (if (L.type_of r1) != sequence_t then
          build_seq_of_int r1 builder else r1)
        and c2 = (if (L.type_of c1) != sequence_t then
          build_seq_of_int c1 builder else c1)
        in build_funcall "massign" [| s1 ; r2 ; c2; v1 |] builder

      | A.Seqassign (s, e, v) -> let e1 = expr builder e and
            s1 = expr builder s and v1 = expr builder v in
          if (L.type_of e1) = i32_t  (* directly put if index e is int *)
          then (ignore(build_funcall "checkseqlength" [| s1; e1 |] builder);
                ignore(build_funcall "checkseqscalar" [| v1 |] builder);
                let v2 = build_get v1 zero_32t builder
                in ignore(build_put s1 e1 v2 builder); v1)
          else build_funcall "vassign" [| s1 ; e1 ; v1 |] builder

      (*---------------------------------------------------------------------
        Type conversion operators
       ---------------------------------------------------------------------*)
      | A.Call ("float_of_int", [e]) ->
          L.build_sitofp (expr builder e) double_t "float_of" builder
      | A.Call ("int_of_float", [e]) ->
          L.build_fptosi (expr builder e) i32_t "int_of" builder
      | A.Call ("int_of_seq", [e]) -> let e1 = (expr builder e) in
        ignore(build_funcall "checkseqscalar" [| e1 |] builder);
        build_get e1 zero_32t builder
      | A.Call ("float_of_mat", [e]) -> let e1 = (expr builder e) in
        ignore(build_funcall "checkmatscalar" [| e1 |] builder);
        build_get e1 zero_32t builder

     (*----------------------------------------------------------------------
       Construct new matrix, sequence, string allocated from stack
      ----------------------------------------------------------------------*)
      | A.Call ("matrix", [e; e1])  ->
          build_matnew select_stack (expr builder e) (expr builder e1) builder
      | A.Call ("sequence", [e]) ->
          build_vecnew select_stack (expr builder e) int_sz builder
      | A.Call ("string", []) ->
          build_new select_stack string_sz builder
      | A.Call ("string", act) ->
          let actuals = Array.of_list (List.map (expr builder) act)
          and s = build_new select_stack string_sz builder in
          ignore (L.build_call snprintf
            (Array.append [| s; string_sz |] actuals) "snpr" builder);
          s
      (*---------------------------------------------------------------------
        Rudimentary output functions, compatible with MicroC
       --------------------------------------------------------------------*)
      | A.Call ("printf", act) ->
          let actuals = List.map (expr builder) act in
          L.build_call printf_func (Array.of_list actuals) "printf" builder
      | A.Call ("print", [e]) ->
          ignore(build_funcall "printint" [| (expr builder e) |] builder);
          build_funcall "println" [| |] builder
      | A.Call ("printb", [e]) ->
          ignore(build_funcall "printbool" [| (expr builder e) |] builder);
          build_funcall "println" [| |] builder

       (*--------------------------------------------------------------------
       Builtin operators length() cols() return size of matrix or sequence
       --------------------------------------------------------------------*)
      | A.Call ("length", [e])  ->
          let null = L.build_is_null (expr builder e) "tmp" builder in
          L.build_select null zero_32t  (* return 0 for null objects *)
            (getdim (expr builder e) length_offset builder) "tmp" builder

      | A.Call ("cols", [e])  ->
          let null = L.build_is_null (expr builder e) "tmp" builder in
          L.build_select null zero_32t  (* return 0 for null objects *)
            (getdim (expr builder e) cols_offset builder) "tmp" builder

      | A.Call (f, act) ->
          let actuals = List.rev (List.map (expr builder) (List.rev act)) in
          if (StringMap.mem f external_decls) then
            build_external f (Array.of_list actuals) builder else
            build_funcall f (Array.of_list actuals) builder
     in

     (* Invoke "f builder" if the current block doesn't already
       have a terminal (e.g., a branch). *)
     let add_terminal builder f =
      match L.block_terminator (L.insertion_block builder) with
       Some _ -> ()
      | None -> ignore (f builder) in

     (* Build the code for the given statement; return the builder for
       the statement's successor *)
     let rec stmt builder = function
       A.Block sl -> List.fold_left stmt builder sl
      | A.Expr e -> ignore (expr builder e); builder

      | A.Return e -> ignore (match fdecl.A.typ with
        (* when return type is mat/seq/str, copy to heap for return *)
        | A.Matrix | A.Sequence | A.String ->
            let e2 = build_copy (expr builder e) select_heap false builder in
            L.build_ret e2 builder
        | A.Void -> L.build_ret_void builder
        | _ -> L.build_ret (expr builder e) builder); builder

      | A.If (predicate, then_stmt, else_stmt) ->
          let bool_val = expr builder predicate in
          let merge_bb = L.append_block context "merge" the_function in

          let then_bb = L.append_block context "then" the_function in
          add_terminal (stmt (L.builder_at_end context then_bb) then_stmt)
            (L.build_br merge_bb);

          let else_bb = L.append_block context "else" the_function in
          add_terminal (stmt (L.builder_at_end context else_bb) else_stmt)
            (L.build_br merge_bb);

          ignore (L.build_cond_br bool_val then_bb else_bb builder);
```

```
          L.builder_at_end context merge_bb

      | A.While (predicate, body) ->
          let pred_bb = L.append_block context "while" the_function in
          ignore (L.build_br pred_bb builder);

          let body_bb = L.append_block context "while_body" the_function in
          add_terminal (stmt (L.builder_at_end context body_bb) body)
            (L.build_br pred_bb);

          let pred_builder = L.builder_at_end context pred_bb in
          let bool_val = expr pred_builder predicate in

          let merge_bb = L.append_block context "merge" the_function in
          ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);
          L.builder_at_end context merge_bb

      | A.For (e1, e2, e3, body) -> stmt builder
          ( A.Block [A.Expr e1 ; A.While (e2, A.Block [body ; A.Expr e3]) ] )
    in

    (* Build the code for each statement in the function *)
    let builder = stmt builder (A.Block fdecl.A.body) in

    (* Add a return if the last block falls off the end *)
    add_terminal builder (match fdecl.A.typ with
      A.Void -> L.build_ret_void
    | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
  in

  List.iter build_function_body local_functions;
  the_module
```

8.1.6 Compiler Top Level Program

Listing 11: Compiler – minimat.ml

```
(* Top-level of the Minimat compiler: scan & parse the input,
   check the resulting AST, generate LLVM IR, and dump the module *)
(* Minimat by Terence Lim tl2735@columbia.edu for COMS4115 *)

type action = Ast | LLVM_IR | Compile

let _ =
  let action = if Array.length Sys.argv > 1 then
    List.assoc Sys.argv.(1) [ ("-a", Ast);      (* Print the AST only *)
                              ("-l", LLVM_IR);  (* Generate LLVM, no check *)
                              ("-c", Compile) ] (* Generate, check LLVM IR *)
  else Compile in
  let lexbuf = Lexing.from_channel stdin in
  let ast = Parser.program Scanner.token lexbuf in
  Semant.check ast;
  match action with
```

```
    Ast -> print_string (Ast.string_of_program ast)
  | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate ast))
  | Compile -> let m = Codegen.translate ast in
    Llvm_analysis.assert_valid_module m;
    print_string (Llvm.string_of_llmodule m)
```

8.2 Support Libraries (coded using MiniMat language)

Much of the syntax of MiniMat is implemented with helper functions coded in MiniMat language itself, which are collected in five library source files:

`expressions.mm`	implement matrix and sequence bracket and colon expressions
`operators.mm`	implement matrix arithmetic and relational operators
`functions.mm`	extensive library of matrix math functions
`io.mm`	input and output functions
`external.mm`	external declarations to use Gnuplot C API for graphical plots

8.2.1 Expressions

Listing 12: Library – expressions.mm

```
/**************************************************************
  expressions.mm -- library helper functions to implement expressions
  Minimat by Terence Lim tl2735@columbia.edu for COMS4115
**************************************************************/
/*-------------------------------------------------------------
  rows(), end(), size()
    -- return dimensional attributes of matrix or sequence object
-------------------------------------------------------------*/
int end(sequence m) { return length(m) - 1; }
int size(matrix m)  { return length(m); }
int rows(matrix m) {
  if (cols(m) == 0) return 0;
  else return length(m) / cols(m);
}

/*-------------------------------------------------------------
  type-conversion functions
-------------------------------------------------------------*/

external int atoi(string s);
external float atof(string s);

float float_of_string(string s) { return atof(s); }
int   int_of_string(string s)   { return atoi(s); }
string string_of_int(int d)     { return new string("%d", d); }
string string_of_float(float f) { return new string("%f", f); }

matrix mat_of_seq(sequence v) {
  matrix a;
  int i;
  a = new matrix(1, length(v));
  for (i = 0; i < length(v); i = i + 1)
    a[0, i] = [float_of_int(int_of_seq(v[i]));];
  return a;
}

sequence seq_of_mat(matrix v) {
  sequence a;
  int i;
  a = new sequence(size(v));
  for (i = 0; i < size(v); i = i + 1)
    a[i] = [int_of_float(float_of_mat(v[i / cols(v), i % cols(v)]))];
  return a;
}

/*------------------------------------------------------------------
  VERTCAT -- helper function to construct matrix expression [...;...;].
    Concatenate columns make taller matrix.
------------------------------------------------------------------*/
matrix vertcat(matrix left, matrix right) {
  matrix out;
  int i;
  int j;

  /* one matrix can be empty, else both must have same number of cols */
  if (!ismatempty(left) && !ismatempty(right)) checkmatcols(left, right);
  out = new matrix(rows(left) + rows(right), maxint2(cols(left),cols(right)));
  for(i = 0; i < cols(left); i = i + 1) {
    for(j = 0; j < rows(left); j = j + 1) {
      out[j, i] = left[j, i];
    }
  }
  for(i = 0; i < cols(right); i = i + 1) {
    for(j = 0; j < rows(right); j = j + 1) {
      out[j + rows(left), i] = right[j, i];
    }
  }
  return out;
}

/*------------------------------------------------------------------
  HORZCAT -- helper function to construct a matrix row expression [1, 2, ...;]
    Concatenate rows make wider matrix
------------------------------------------------------------------*/
matrix horzcat(matrix left, matrix right) {
  matrix out;
  int i;
  int j;

  /* one matrix can be empty, else both must have same number of rows */
  if (!ismatempty(left) && !ismatempty(right)) checkmatrows(left, right);
  out = new matrix(maxint2(rows(left),rows(right)), cols(left) + cols(right));
  for(i = 0; i < rows(left); i = i + 1) {
    for(j = 0; j < cols(left); j = j + 1) {
      out[i, j] = left[i, j];
    }
  }
  for(i = 0; i < rows(right); i = i + 1) {
    for(j = 0; j < cols(right); j = j + 1) {
      out[i, j + cols(left)] = right[i, j];
    }
  }
  return out;
}

/*---------------------------------------------------------
  MSELECT -- helper function for matrix subselect expression A[2,d]
---------------------------------------------------------*/
matrix mselect(matrix right, sequence row, sequence col) {
  matrix left;
  int i;
  int j;
  left = new matrix(length(row), length(col));
  for(i = 0; i < length(row); i = i + 1) {
    for(j = 0; j < length(col); j = j + 1) {
      checkmatrc(right, int_of_seq(row[i]), int_of_seq(col[j]));
      left[i, j] = right[int_of_seq(row[i]), int_of_seq(col[j])];
    }
  }
  return left;
}

/*---------------------------------------------------------
   MASSIGN -- helper function for matrix subassignment expression A[1,d] =
---------------------------------------------------------*/
matrix massign(matrix left, sequence row, sequence col, matrix right) {
  int i;
  int j;
  for(i = 0; i < length(row); i = i + 1) {
    for(j = 0; j < length(col); j = j + 1) {
      checkmatrc(left, int_of_seq(row[i]), int_of_seq(col[j]));
      if (cols(right) == 1 && rows(right) == 1) {
        left[int_of_seq(row[i]), int_of_seq(col[j])] = right[0, 0];
      }
      else {
        checkmatrc(right, i, j);
        left[int_of_seq(row[i]), int_of_seq(col[j])] = right[i, j];
      }
    }
  }
  return right;
}

/*---------------------------------------------------------
  select and assign to matrix with sequence of linear-method indexes
---------------------------------------------------------*/
/* select right[s] */
```

```
matrix mat_select_seq(matrix right, sequence s) {
  matrix x;
  int i;
  x = new matrix(1, length(s));
  for (i = 0; i < length(s); i = i + 1) {
    checkmatindex(right, int_of_seq(s[i]));
    x[0, i] = right[int_of_seq(s[i]) / cols(right),
                    int_of_seq(s[i]) % cols(right)];
  }
  return x;
}

/* assign left[s] = right */
matrix mat_assign_seq(matrix left, sequence s, matrix right) {
  int i;
  matrix x;
  if (!ismatscalar(right)) checkmatsize(left,right);
  for (i = 0; i < length(s); i = i + 1) {
    checkmatindex(left, int_of_seq(s[i]));
    if (ismatscalar(right)) x = right;
    else x = right[i / cols(right), i % cols(right)];
    left[int_of_seq(s[i]) / cols(left), int_of_seq(s[i]) % cols(left)] = x;
  }
  return right;
}

void checkmatindex(matrix v, int i) {
  if (i >= size(v) || i < 0) errorexit("matrix linear index out of bounds");
}

/*-------------------------------------------------------------
  APPEND -- define helper function to construct sequence literal [1, v, 5]
-------------------------------------------------------------*/
sequence append(sequence left, sequence right) {
  sequence out;
  int i;
  out = new sequence(length(left) + length(right));
  for(i = 0; i < length(left); i = i + 1) {
    out[i] = left[i];
  }
  for(i = 0; i < length(right); i = i + 1) {
    out[i + length(left)] = right[i];
  }
  return out;
}

/*-------------------------------------------------------------
   VSELECT -- define helper for sequence subselect expression V[d]
-------------------------------------------------------------*/
sequence vselect(sequence right, sequence select) {
  sequence left;
  int i;
  int j;
  left = new sequence(length(select));
  for(i = 0; i < length(select); i = i + 1) {
    j = int_of_seq(select[i]);
    checkseqlength(right, j);
    left[i] = right[j];
  }
  return left;
}

/*------------------------------------------------------------
    VASSIGN -- define helper for sequence subassignment expression A(v) =
------------------------------------------------------------*/
sequence vassign(sequence left, sequence select, sequence right) {
  int i;
  int j;
  for(i = 0; i < length(select); i = i + 1) {
    checkseqlength(left, int_of_seq(select[i]));
    if (length(select) == 1) j = 0; else j = i;
    checkseqlength(right,j);
    left[int_of_seq(select[i])] =  right[j];
  }
  return right;
}

/*------------------------------------------------------------
  STRIDE -- define helper for colon expression 1::3 10:5:30
------------------------------------------------------------*/
sequence stride(int beg, int by, int end) {
  int n;
  int i;
  sequence v;
  if ((beg <= end && by > 0) || (beg > end && by < 0)) {
    end = beg+(by*((end - beg) / by));
    n = ((end - beg)/by) + 1;
  }
  else {
    n = 0;
  }
  v = new sequence(n);
  for(i = 0; i < n; i = i + 1) {
    v[i] = [beg + (i * by)];
  }
  return v;
}

/*------------------------------------------------------------
  Error exit function
------------------------------------------------------------*/
external void exit(int i);
void errorexit(string s) { printf("%s.  Exiting...\n",s); exit(0);}

/*------------------------------------------------------------
    utility functions to check dimensions
------------------------------------------------------------*/

```

```c
/*  Errorexit if row and column index out of matrix bounds */

void checkmatrc(matrix v, int i, int j) {
  if (i >= rows(v) || i < 0 || j >= cols(v) || j < 0)
    errorexit("matrix row-column index out of bounds");
}

/* Errorexit if index position out of sequence bounds */
void checkseqlength(sequence v, int i) {
  if (i >= length(v) || i < 0) errorexit("sequence index out of bounds");
}

/* Errorexit if matrix is empty */
void checkmatempty(matrix u) {
  if (cols(u) == 0 || rows(u) == 0)
    errorexit("Matrix cannot be zero length");
}

/* Returns true if matrix is empty */
bool ismatempty(matrix u) { return (cols(u) == 0 || rows(u) == 0); }

/* Errorexit if two matrices have different number of columns */
void checkmatcols(matrix u, matrix v) {
  if (cols(u) != cols(v))
    errorexit("Matrices cannot have different col size");
}

/* Errorexit if two matrices have different number of rows */
void checkmatrows(matrix u, matrix v) {
  if (rows(u) != rows(v))
    errorexit("Matrices cannot have different row size");
}

/* Errorexit if two matrices have different dimensions */
void checkmatdims(matrix u, matrix v) {
  checkmatcols(u, v);
  checkmatrows(u, v);
}

/* Errorexit if two matrices have different capacity */
void checkmatsize(matrix u, matrix v) {
  if (size(u) != size(v))
    errorexit("Matrices cannot have different capacity");
}

/* Returns true if matrix is singleton */
bool ismatscalar(matrix u) { return rows(u) == 1 && cols(u) == 1; }

/* Errorexit if matrix is singleton */
void checkmatscalar(matrix u) {
  if (!ismatscalar(u)) errorexit("matrix not a scalar");
}

void checkseqscalar(sequence v) {
  if (length(v) != 1) errorexit("sequence not a scalar");
}

/* Errorexit if matrix is not square */
void checkmatsquare(matrix u) {
  if (cols(u) != rows(u)) errorexit("matrix is not square");
}

/* Errorexit if sequence is empty */
void checkseqempty(sequence u) {
  if (length(u) == 0) errorexit("Sequence cannot be zero length");
}

/* Errorexit if two sequences have different length */
void checkseqsize(sequence u, sequence v) {
  if (length(u) != length(v)) errorexit("Sequences not of same length");
}
```

8.2.2 Operators

Listing 13: Library – operators.mm

```
/************************************************************************
  operators.mm: library of helper functions to implement operators
  Minimat by Terence Lim tl2735@columbia.edu for COMS4115
************************************************************************/
/*-------------------------------------------------------------
  MATBINOP -- helper function for matrix arithmetic operator functions
-------------------------------------------------------------*/
constant int MATADD = 0;
constant int MATSUB = 1;
constant int MATDOTMUL = 2;
constant int MATDOTDIV = 3;
constant int MATDOTREM = 4;
constant int MATDOTPOW = 5;

matrix matbinop(matrix u, matrix v, int t) {
  matrix w;
  float i;
  float j;
  int k;
  int m;
  float x;
  checkmatempty(u);
  checkmatempty(v);
  if (!ismatscalar(u) && !ismatscalar(v)) checkmatsize(u,v);
  w = new matrix(maxint2(rows(u),rows(v)), maxint2(cols(v),cols(u)));
  for (k = 0; k < rows(w); k = k + 1) {
    for (m = 0; m < cols(w); m = m + 1) {
      if (ismatscalar(u)) i = float_of_mat(u[0, 0]);
      else i = float_of_mat(u[k, m]);
      if (ismatscalar(v)) j = float_of_mat(v[0, 0]);
      else j = float_of_mat(v[k, m]);
      if (t == MATADD) x = i + j;
      else if (t == MATSUB) x = i - j;
      else if (t == MATDOTMUL) x = i * j;
      else if (t == MATDOTDIV) x = i / j;
      else if (t == MATDOTREM) x = i % j;
      else if (t == MATDOTPOW) x = i ^ j;
      else errorexit("illegal matrix binop");
      w[k, m] = [x;];
    }
  }
  return w;
}

/*----------------------------------------------------------------
  Define helper functions for matrix binary operators: + -
----------------------------------------------------------------*/
matrix madd(matrix u, matrix v)    { return matbinop(u, v, MATADD); }
matrix msub(matrix u, matrix v)    { return matbinop(u, v, MATSUB); }
matrix mdotmul(matrix u, matrix v) { return matbinop(u, v, MATDOTMUL); }
matrix mdotdiv(matrix u, matrix v) { return matbinop(u, v, MATDOTDIV); }
matrix mdotrem(matrix u, matrix v) { return matbinop(u, v, MATDOTREM); }
matrix mdotpow(matrix u, matrix v) { return matbinop(u, v, MATDOTPOW); }

/*----------------------------------------------------------------
  MATBINCOMP -- helper function for matrix comparison operators
----------------------------------------------------------------*/
constant int MATLT = 10;
constant int MATLE = 11;
constant int MATGT = 12;
constant int MATGE = 14;
constant int MATEQ = 15;
constant int MATNE = 16;

sequence matbincomp(matrix u, matrix v, int t) {
  sequence w;
  sequence x;
  int n;
  float i;
  float j;
  int k;
  int m;
  int h;
  bool b;
  int r;
  int c;
  checkmatempty(u);
  checkmatempty(v);
  if (!ismatscalar(u) && !ismatscalar(v)) checkmatsize(u,v);
  r = maxint2(rows(u), rows(v));
  c = maxint2(cols(v), cols(u));
  w = new sequence(r * c);
  n = 0;
  for (k = 0; k < r; k = k + 1) {
    for (m = 0; m < c; m = m + 1) {
      if (ismatscalar(u)) i = float_of_mat(u[0, 0]);
      else i = float_of_mat(u[k, m]);
      if (ismatscalar(v)) j = float_of_mat(v[0, 0]);
      else j = float_of_mat(v[k, m]);
      if (t == MATLT) b = i < j;
      else if (t == MATLE) b = i <= j;
      else if (t == MATGT) b = i > j;
      else if (t == MATGE) b = i >= j;
      else if (t == MATEQ) b = i == j;
      else if (t == MATNE) b = i != j;
      else errorexit("illegal matrix comparison op");
      if (b) h = 1; else h = 0;
      w[(k * c) + m] = [h];
      n = n + h;
    }
  }
  x = new sequence(n);
  h = 0;
  for (k = 0; k < length(w); k = k + 1) {
    if (int_of_seq(w[k]) == 1) {
      x[h] = [k];
      h = h + 1;
    }
  }
  return x;
}

/*-------------------------------------------------------------------
  Define helper functions for matrix comparison operators: < <= > >= == !=
-------------------------------------------------------------------*/
sequence mlt(matrix u, matrix v) { return matbincomp(u, v, MATLT); }
sequence mle(matrix u, matrix v) { return matbincomp(u, v, MATLE); }
sequence mgt(matrix u, matrix v) { return matbincomp(u, v, MATGT); }
sequence mge(matrix u, matrix v) { return matbincomp(u, v, MATGE); }
sequence meq(matrix u, matrix v) { return matbincomp(u, v, MATEQ); }
sequence mne(matrix u, matrix v) { return matbincomp(u, v, MATNE); }

/*-------------------------------------------------------------------
  MATTRANSP -- define function for "'" matrix postfix transpose operator
-------------------------------------------------------------------*/
matrix mtransp(matrix right) {
  int i;
  int j;
  matrix left;
  left = new matrix(cols(right),rows(right));
  for(i = 0; i < rows(right); i = i + 1) {
    for(j = 0; j < cols(right); j = j + 1) {
      left[j, i] = right[i, j];
    }
  }
  return left;
}

/*-------------------------------------------------------------------
```

```
  MNEG -- define function function for "-" matrix unary prefix operator
-----------------------------------------------------------------*/
matrix mneg(matrix right) {
  matrix left;
  int i;
  int j;
  left = new matrix(rows(right),cols(right));
  for(i = 0; i < rows(right); i = i + 1) {
    for(j = 0; j < cols(right); j = j + 1) {
      left[i, j] = [-float_of_mat(right[i, j]);];
    }
  }
  return left;
}

/*-----------------------------------------------------------------
  MMUL -- define function for the "*" matrix multiply infix operator
-----------------------------------------------------------------*/
matrix mmul(matrix left, matrix right) {
  int i;
  int j;
  int k;
  matrix prod;
  float x;

  if (cols(left) == 1 && rows(left) == 1) {
    prod = new matrix(rows(right), cols(right));
    for(i = 0; i < rows(right); i = i + 1) {
      for(j = 0; j < cols(right); j = j + 1) {
        prod[i, j] = [float_of_mat(left[0, 0]) * float_of_mat(right[i, j]);];
      }
    }
  }
  else if (cols(right) == 1 && rows(right) == 1) {
    prod = new matrix(rows(left), cols(left));
    for(i = 0; i < rows(left); i = i + 1) {
      for(j = 0; j < cols(left); j = j + 1) {
        prod[i, j] = [float_of_mat(left[i, j]) * float_of_mat(right[0, 0]);];
      }
    }
  }
  else if (cols(left) != rows(right)) {
    errorexit("illegal matrix dimensions for multiplication");
  }
  else {
    prod = new matrix(rows(left), cols(right));
    for(i = 0; i < rows(prod); i = i + 1) {
      for(j = 0; j < cols(prod); j = j + 1) {
        x = 0.0;
        for(k = 0; k < cols(left); k = k + 1) {
          x = x + float_of_mat(left[i, k]) * float_of_mat(right[k, j]);
        }
        prod[i, j] = [x;];
      }
    }
  }
  return prod;
}

/*---------------------------------------------------------------------------
  MPOW -- define function for matrix power infix operator "^"
---------------------------------------------------------------------------*/
matrix mpow(matrix u, int k) {
  matrix w;
  int i;
  checkmatsquare(u);
  w = u;
  for(i = 1; i < k; i = i + 1) w = mmul(w,u);
  return w;
}
/**/
/*---------------------------------------------------------------------------
  MDIV -- define function for matrix divide infix operator "/"
---------------------------------------------------------------------------*/
matrix mdiv(matrix y, matrix x) {
  checkmatrows(y,x);
  return inv(x' * x) * (x' * y);
}
/**/
/*---------------------------------------------------------------------------
  MREM -- define function for matrix remainder infix operator "/"
---------------------------------------------------------------------------*/
matrix mrem(matrix y, matrix x) {
  matrix b;
  b = mdiv(y,x);
  return y - (x * b);
}

/*---------------------------------------------------------------------------
  VECBINOP -- helper function for sequence arithmetic operator functions
---------------------------------------------------------------------------*/
constant int VECADD = 20;
constant int VECSUB = 21;
constant int VECMUL = 22;
constant int VECDIV = 23;
constant int VECREM = 24;

sequence vecbinop(sequence u, sequence v, int t) {
  sequence w;
  int u1;
  int v1;
  int i;
  int ans;
  checkseqempty(u);
  checkseqempty(v);
  if (length(u) > 1 && length(v) > 1) checkseqsize(u,v);
  w = new sequence(maxint2(length(u), length(v)));
  for (i = 0; i < length(w); i = i+1) {
    if (length(u) == 1) u1 = int_of_seq(u[0]); else u1 = int_of_seq(u[i]);
    if (length(v) == 1) v1 = int_of_seq(v[0]); else v1 = int_of_seq(v[i]);
    if (t == VECADD) ans = u1 + v1;
    else if (t == VECSUB) ans = u1 - v1;
    else if (t == VECMUL) ans = u1 * v1;
    else if (t == VECDIV) ans = u1 / v1;
    else if (t == VECREM) ans = u1 % v1;
    else errorexit("illegal sequence binop");
    w[i] = [ans];
  }
  return w;
}

/*---------------------------------------------------------------------------
  Define helper functions for sequence binary arithmetic operators: + - * / %
---------------------------------------------------------------------------*/
sequence vadd(sequence u, sequence v) { return vecbinop(u, v, VECADD); }
sequence vsub(sequence u, sequence v) { return vecbinop(u, v, VECSUB); }
sequence vmul(sequence u, sequence v) { return vecbinop(u, v, VECMUL); }
sequence vdiv(sequence u, sequence v) { return vecbinop(u, v, VECDIV); }
sequence vrem(sequence u, sequence v) { return vecbinop(u, v, VECREM); }

/*---------------------------------------------------------------------------
  Define helper function for the "-" sequence unary prefix operator
---------------------------------------------------------------------------*/
sequence vneg(sequence right) {
  int i;
  sequence left;
  left = new sequence(length(right));
  for(i = 0; i < length(right); i = i + 1) {
    left[i] = [-int_of_seq(right[i])];
  }
  return left;
}


/*---------------------------------------------------------------------------
  STRINGEQ STRINGNE -- define functions string comparison operators: == !=
---------------------------------------------------------------------------*/
external int strcmp(string s, string t);
bool stringeq(string a, string b) { return (strcmp(a,b) == 0); }
bool stringne(string a, string b) { return (strcmp(a,b) != 0); }
bool stringge(string a, string b) { return (strcmp(a,b) >= 0); }
bool stringgt(string a, string b) { return (strcmp(a,b) >  0); }
bool stringle(string a, string b) { return (strcmp(a,b) <= 0); }
bool stringlt(string a, string b) { return (strcmp(a,b) <  0); }

/* returns max or min of two ints */
int maxint2(int i, int j) { if (i > j) return i; else return j; }
int minint2(int i, int j) { if (i < j) return i; else return j; }
```

8.2.3 Functions

Listing 14: Library – functions.mm

```
/**************************************************************************
   functions.mm: library of matrix functions
   Minimat by Terence Lim tl2735@columbia.edu for COMS4115
**************************************************************************/
/*------------------------------------------------------------------------
   declare useful external floating functions
------------------------------------------------------------------------*/
external float fabs(float x);
external float exp(float x);
external float log(float x);
external float pow(float x, float y);
float sqrt(float x) { return pow(x,0.5); }

/*------------------------------------------------------------------------
  MEXP MLOG MABS -- applies unary math function on each matrix element
------------------------------------------------------------------------*/
constant int MATEXP = 1;
constant int MATLOG = 2;
constant int MATABS = 3;

/* helper for unary matrix math functions */
matrix matuop(matrix x, int op) {
  matrix y;
  int i;
  int j;
  float z;
  y = new matrix(rows(x), cols(x));
  for (i = 0; i < rows(y); i = i + 1) {
    for (j = 0; j < cols(y); j = j + 1) {
      z = float_of_mat(x[i, j]);
      if (op == MATEXP) z = exp(z);
      else if (op == MATLOG) z = log(z);
      else if (op == MATABS) z = fabs(z);
      else errorexit("illegal matrix uop");
      y[i, j] = [z;];
    }
  }
  return y;
}
matrix mexp(matrix x) { return matuop(x, MATEXP); }
matrix mlog(matrix x) { return matuop(x, MATLOG); }
matrix mabs(matrix x) { return matuop(x, MATABS); }

/*------------------------------------------------------------------------
  EYE -- constructs an identity matrix
------------------------------------------------------------------------*/
matrix eye(int n) {
  matrix x;
  x = new matrix(n, n);
  mat_assign_seq(x,0 : cols(x)+1 : size(x)-1, [1.0;]);
  return x;
}
```

```

/*---------------------------------------------------------------------------
  DIAG -- extracts diagonal elements from a matrix
---------------------------------------------------------------------------*/
matrix diag(matrix x) { return mat_select_seq(x, 0 : cols(x)+1 : size(x)-1); }

/*---------------------------------------------------------------------------
  ONES -- constructs a matrix of 1's
---------------------------------------------------------------------------*/
matrix ones(int m, int n) { return new matrix(m, n) + [1.0;]; }

/*---------------------------------------------------------------------------
  RESHAPE -- reshapes a matrix to new dimensions
---------------------------------------------------------------------------*/
matrix reshape(matrix a, int r, int c) {
  matrix b;
  b = new matrix(r, c);
  checkmatsize(a,b);
  mat_assign_seq(b, 0::size(b) - 1, a);
  return b;
}

/*---------------------------------------------------------------------------
  SUM -- sums matrix elements and returns as a float
---------------------------------------------------------------------------*/
float sum(matrix x) {
  float sum;
  int i;
  int j;
  sum = 0.0;
  for (i = 0; i < rows(x); i = i + 1) {
    for (j = 0; j < cols(x); j = j + 1) {
      sum = sum + float_of_mat(x[i, j]);
    }
  }
  return sum;
}

/*---------------------------------------------------------------------------
  MEAN -- returns average value of matrix elements
---------------------------------------------------------------------------*/
float mean(matrix x) { return sum(x) / float_of_int(size(x)); }

/*---------------------------------------------------------------------------
  NORM -- returns euclidean L2-norm of matrix values
---------------------------------------------------------------------------*/
float norm(matrix x) { return sqrt(sum(x .^ [2.0;])); }

/*---------------------------------------------------------------------------
  MIN -- returns minimum value in matrix
---------------------------------------------------------------------------*/
float min(matrix x) {
  int i;
  int j;
  float min;
  float tmp;
  min = float_of_mat(x[0,0]);
  for (i = 0; i < rows(x); i = i + 1) {
    for (j = 0; j < cols(x); j = j + 1) {
      tmp = float_of_mat(x[i, j]);
      if (tmp < min) min = tmp;
    }
  }
  return min;
}

/*--------------------------------------------------------------
  MAX -- returns maximum value in matrix
--------------------------------------------------------------*/
float max(matrix x) {
  int i;
  int j;
  float max;
  float tmp;
  max = float_of_mat(x[0,0]);
  for (i = 0; i < rows(x); i = i + 1) {
    for (j = 0; j < cols(x); j = j + 1) {
      tmp = float_of_mat(x[i, j]);
      if (tmp > max) max = tmp;
    }
  }
  return max;
}

/*--------------------------------------------------------------
  TRIL -- returns lower triangular submatrix
--------------------------------------------------------------*/
matrix tril(matrix a, int k) {
  matrix b;
  int r;
  int c;
  b = a;
  for (r = 0; r < rows(a); r = r + 1) {
    for (c = r + 1 + k; c < cols(a); c = c + 1) {
      b[r, c] = [0.0;];
    }
  }
  return b;
}

/*--------------------------------------------------------------
  TRIU -- returns upper triangular submatrix
--------------------------------------------------------------*/
matrix triu(matrix a, int k) {
  matrix b;
  int r;
  int c;
  b = new matrix(rows(a), cols(a));
  for (r = 0; r < rows(a); r = r + 1) {
    for (c = r + k; c < cols(a); c = c + 1) {
      b[r, c] = a[r, c];
    }
  }
  return b;
}
/**/
/*---------------------------------------------------------------------
  DET -- computes determinant by recursively expanding minors
---------------------------------------------------------------------*/
float det(matrix a) {
  matrix det;
  int i;
  int j;
  int j1;
  int j2;
  matrix m;
  float tmp;
  checkmatsquare(a);
  if (rows(a) == 1) det = a[0, 0];
  else if (rows(a) == 2) det = a[0, 0] * a[1, 1] - a[0, 1] * a[1, 0];
  else {
    det = [0.0;];
    for (j1 = 0; j1 < cols(a); j1 = j1 + 1) {
      m = new matrix(rows(a) - 1, cols(a) - 1);
      for (i = 1; i < rows(a); i = i + 1) {
        j2 = 0;
        for (j = 0; j < cols(a); j = j + 1) {
            if (j != j1) {
              m[i-1, j2] = a[i, j];
              j2 = j2 + 1;
            }
        }
      }
      det = det + [(-1.0 ^ (float_of_int(j1) + 2.0));] * a[0, j1] * [det(m);];
    }
  }
  return float_of_mat(det);
}

/*---------------------------------------------------------------------
  COFACTOR -- returns cofactor of a matrix
---------------------------------------------------------------------*/
matrix cofactor(matrix a) {
  int i;
  int j;
  int ii;
  int jj;
  int i1;
  int j1;
  float det;
  matrix c;
  int n;
  matrix b;
  checkmatsquare(a);
  n = rows(a);
  b = new matrix(n, n);
  c = new matrix(n-1, n-1);
  for (j = 0; j < n; j = j + 1) {
    for (i = 0; i < n; i = i + 1) {
      i1 = 0;
      for (ii = 0; ii < n; ii = ii + 1) {
        if (ii != i) {
          j1 = 0;
          for (jj = 0; jj < n; jj = jj + 1) {
            if (jj != j) {
              c[i1, j1] = a[ii, jj];
              j1 = j1 + 1;
            }
          }
          i1 = i1 + 1;
        }
      }
      b[i, j] = [(-1.0 ^ (float_of_int(i+j)+2.0)) * det(c);];
    }
  }
  return b;
}

/*------------------------------------------------------------
  INV -- returns inverse of matrix
------------------------------------------------------------*/
matrix inv(matrix a) { return cofactor(a)' ./ [det(a);]; }
/**/
/*------------------------------------------------------------
  ADJOINT -- returns adjoint of matrix
------------------------------------------------------------*/
matrix adjoint(matrix a) { return cofactor(a)'; }

/*------------------------------------------------------------
  REGRESS -- displays regression fit, returns predicted values
------------------------------------------------------------*/
matrix regress(matrix y, matrix x) {
  matrix b;
  matrix se;
  matrix yhat;
  x = [ones(rows(x),1), x;];
  b = y / x;
  yhat = x * b;
  se = ([norm(y - yhat);] * (diag(inv(x' * x)) .^ [0.5;]))
         ./ [sqrt(float_of_int(size(yhat)));];
  printmat(b');
  printmat(se);
  printmat(b' ./ se);
  return yhat;
}
```

8.2.4 Input/Output

Listing 15: Library – io.mm

```
/************************************************
  io.mm -- basic i/o and type conversion functions
  Minimat by Terence Lim tl2735@columbia.edu for COMS4115
************************************************/
/*-----------------------------------------------
  Define basic print to stdout functions
-----------------------------------------------*/
void println()            { printf("\n"); }
void printint(int i)      { printf("%d ",i); }
void printbool(bool b)    { if (b) printint(1); else printint(0); }
void printfloat(float f)  { printf("%6.2f ",f); }
void printstring(string s) { printf("%s ",s); }
void printhandle(handle i) { printf("%p ", i); }
void printdims(matrix x)  { printf("%d %d\n",rows(x),cols(x)); }

void printseq(sequence v) {
  int n;
  int i;
  n = length(v);
  printf("[%d int]\n",n);
  for(i = 0; i < n; i = i + 1) printint(int_of_seq(v[i]));
  if (i > 0) println();
}

void printmat(matrix v) {
  int c;
  int r;
  int i;
  int j;
  c = cols(v);
  r = rows(v);
  printf("[%d x %d float]\n",r,c);
  for(i = 0; i < r; i = i + 1) {
    for(j = 0; j < c; j = j + 1) {
      printfloat(float_of_mat(v[i, j]));
    }
    println();
  }
}

/*-----------------------------------------------
  Define basic input from stdin functions
-----------------------------------------------*/
external int scanf(string s, string h);
string next() {
  string h;
  h = new string();
  scanf("%255s",h);
  return h;
}
```

```

float nextfloat() { return float_of_string(next()); }
int   nextint()   { return int_of_string(next()); }
```

8.2.5 Externals

Listing 16: Library – external.mm

```
/*************************************************
  external.mm -- external library routines
    GNUPLOT for visualizing plots (C API by N. Devillard)
  Minimat by Terence Lim tl2735@columbia.edu for COMS4115
*************************************************/
/**/
/* Declare external GNUPLOT C API -- for visualizing plots */
external handle gnuplot_init();
external void gnuplot_cmd(handle g, string c);
external void gnuplot_plot_equation(handle g, string c,string s);
external void gnuplot_close(handle g);
external void gnuplot_plot_xy(handle g, matrix x, matrix y, int n, string s);
external void gnuplot_setstyle(handle g, string s);
  /* lines points linespoints impulses dots steps errorbars boxes */
external void gnuplot_resetplot(handle g);
external void gnuplot_set_xlabel(handle g, string s);
external void gnuplot_set_ylabel(handle g, string s);

/* sets output to a PNG picture file */
void gnuplot_set_png(handle g, string f) {
  gnuplot_cmd(g, "set terminal png");
  gnuplot_cmd(g, new string("set output \"%s\"", f));
}

/* sets yrange of plot from min and max values of data set */
void gnuplot_set_yrange(handle g, matrix y) {
  gnuplot_cmd(g, new string("set yrange [%g:%g]", min(y), max(y)));
}

/* sets xrange of plot from min and max values of data set */
void gnuplot_set_xrange(handle g, matrix x) {
  gnuplot_cmd(g, new string("set xrange [%g:%g]", min(x), max(x)));
}
/**/
```

8.3 Target LLVM Generated Code of Sample Programs

8.3.1 Sample Program I: Matrix Inverse and Linear Regression

Listing 17: Sample target code – regression.ll

```
; ModuleID = 'MiniMat'
```

```
@_null = global [1 x i8] zeroinitializer
@str = private unnamed_addr constant [21 x i8] c"Number of outliers: \00"
@str1 = private unnamed_addr constant [6 x i8] c"%255s\00"
@str2 = private unnamed_addr constant [17 x i8] c"[%d x %d float]\0A\00"
@str3 = private unnamed_addr constant [10 x i8] c"[%d int]\0A\00"
@str4 = private unnamed_addr constant [7 x i8] c"%d %d\0A\00"
@str5 = private unnamed_addr constant [4 x i8] c"%p \00"
@str6 = private unnamed_addr constant [4 x i8] c"%s \00"
@str7 = private unnamed_addr constant [7 x i8] c"%6.2f \00"
@str8 = private unnamed_addr constant [4 x i8] c"%d \00"
@str9 = private unnamed_addr constant [2 x i8] c"\0A\00"
@str10 = private unnamed_addr constant [19 x i8] c"illegal matrix uop\00"
@str11 = private unnamed_addr constant [23 x i8] c"illegal sequence binop\00"
@str12 = private unnamed_addr constant [45 x i8] c"illegal matrix dimensions for
    multiplication\00"
@str13 = private unnamed_addr constant [29 x i8] c"illegal matrix comparison op\00"
@str14 = private unnamed_addr constant [21 x i8] c"illegal matrix binop\00"
@str15 = private unnamed_addr constant [29 x i8] c"Sequences not of same length\00"
@str16 = private unnamed_addr constant [31 x i8] c"Sequence cannot be zero length\00"
@str17 = private unnamed_addr constant [21 x i8] c"matrix is not square\00"
@str18 = private unnamed_addr constant [22 x i8] c"sequence not a scalar\00"
@str19 = private unnamed_addr constant [20 x i8] c"matrix not a scalar\00"
@str20 = private unnamed_addr constant [40 x i8] c"Matrices cannot have different
    capacity\00"
@str21 = private unnamed_addr constant [40 x i8] c"Matrices cannot have different row
    size\00"
@str22 = private unnamed_addr constant [40 x i8] c"Matrices cannot have different col
    size\00"
@str23 = private unnamed_addr constant [29 x i8] c"Matrix cannot be zero length\00"
@str24 = private unnamed_addr constant [29 x i8] c"sequence index out of bounds\00"
@str25 = private unnamed_addr constant [38 x i8] c"matrix row-column index out of
    bounds\00"
@str26 = private unnamed_addr constant [17 x i8] c"%s. Exiting...\0A\00"
@str27 = private unnamed_addr constant [34 x i8] c"matrix linear index out of bounds\00"
@str28 = private unnamed_addr constant [3 x i8] c"%f\00"
@str29 = private unnamed_addr constant [3 x i8] c"%d\00"

declare i32 @printf(i8*, ...)

declare i32 @snprintf(i8*, ...)

declare void @memset(i8*, i32, i32)

declare i32 @memcpy(i8*, i8*, i32)

declare i32 @scanf(i8*, i8*)

declare double @pow(double, double)

declare double @log(double)

declare double @exp(double)

declare double @fabs(double)

declare i32 @strcmp(i8*, i8*)

declare void @exit(i32)

declare double @atof(i8*)
declare i32 @atoi(i8*)

define i32 @main() {
entry:
  %y = alloca double*
  store double* null, double** %y
  %x = alloca double*
  store double* null, double** %x
  %xx = alloca double*
  store double* null, double** %xx
  %outliers = alloca i32*
  store i32* null, i32** %outliers
  %new = alloca i8, i32 16
  call void @memset(i8* %new, i32 0, i32 16)
  %new1 = getelementptr i8* %new, i8 16
  %dim = bitcast i8* %new1 to i32*
  %dim2 = getelementptr i32* %dim, i32 -1
  store i32 16, i32* %dim2
  %dim3 = bitcast i8* %new1 to i32*
  %dim4 = getelementptr i32* %dim3, i32 -2
  store i32 0, i32* %dim4
  %dim5 = bitcast i8* %new1 to i32*
  %dim6 = getelementptr i32* %dim5, i32 -4
  store i32 0, i32* %dim6
  %dim7 = bitcast i8* %new1 to i32*
  %dim8 = getelementptr i32* %dim7, i32 -3
  store i32 0, i32* %dim8
  %new9 = bitcast i8* %new1 to i32*
  %dim10 = getelementptr i32* %new9, i32 -4
  store i32 0, i32* %dim10
  %dim11 = getelementptr i32* %new9, i32 -3
  store i32 0, i32* %dim11
  %new12 = bitcast i32* %new9 to double*
  %new13 = alloca i8, i32 24
  call void @memset(i8* %new13, i32 0, i32 24)
  %new14 = getelementptr i8* %new13, i8 16
  %dim15 = bitcast i8* %new14 to i32*
  %dim16 = getelementptr i32* %dim15, i32 -1
  store i32 24, i32* %dim16
  %dim17 = bitcast i8* %new14 to i32*
  %dim18 = getelementptr i32* %dim17, i32 -2
  store i32 1, i32* %dim18
  %dim19 = bitcast i8* %new14 to i32*
  %dim20 = getelementptr i32* %dim19, i32 -4
  store i32 0, i32* %dim20
  %dim21 = bitcast i8* %new14 to i32*
  %dim22 = getelementptr i32* %dim21, i32 -3
  store i32 0, i32* %dim22
  %new23 = bitcast i8* %new14 to i32*
  %dim24 = getelementptr i32* %new23, i32 -4
  store i32 1, i32* %dim24
  %dim25 = getelementptr i32* %new23, i32 -3
  store i32 1, i32* %dim25
  %new26 = bitcast i32* %new23 to double*
  %put = getelementptr double* %new26, i32 0
  store double 7.000000e+00, double* %put
  %horzcat_res = call double* @horzcat(double* %new12, double* %new26)
  %dim27 = bitcast double* %horzcat_res to i32*
  %dim28 = getelementptr i32* %dim27, i32 -1
  %dim29 = load i32* %dim28
  %new30 = alloca i8, i32 %dim29
```

```
call void @memset(i8* %new30, i32 0, i32 %dim29)
%new31 = bitcast double* %horzcat_res to i8*
%new32 = getelementptr i8* %new31, i8 -16
%0 = call i32 @memcpy(i8* %new30, i8* %new32, i32 %dim29)
tail call void @free(i8* %new32)
%new33 = getelementptr i8* %new30, i8 16
%cp = bitcast i8* %new33 to double*
%new34 = alloca i8, i32 24
call void @memset(i8* %new34, i32 0, i32 24)
%new35 = getelementptr i8* %new34, i8 16
%dim36 = bitcast i8* %new35 to i32*
%dim37 = getelementptr i32* %dim36, i32 -1
store i32 24, i32* %dim37
%dim38 = bitcast i8* %new35 to i32*
%dim39 = getelementptr i32* %dim38, i32 -2
store i32 1, i32* %dim39
%dim40 = bitcast i8* %new35 to i32*
%dim41 = getelementptr i32* %dim40, i32 -4
store i32 0, i32* %dim41
%dim42 = bitcast i8* %new35 to i32*
%dim43 = getelementptr i32* %dim42, i32 -3
store i32 0, i32* %dim43
%new44 = bitcast i8* %new35 to i32*
%dim45 = getelementptr i32* %new44, i32 -4
store i32 1, i32* %dim45
%dim46 = getelementptr i32* %new44, i32 -3
store i32 1, i32* %dim46
%new47 = bitcast i32* %new44 to double*
%put48 = getelementptr double* %new47, i32 0
store double 7.000000e+00, double* %put48
%horzcat_res49 = call double* @horzcat(double* %new12, double* %new47)
%dim50 = bitcast double* %horzcat_res49 to i32*
%dim51 = getelementptr i32* %dim50, i32 -1
%dim52 = load i32* %dim51
%new53 = alloca i8, i32 %dim52
call void @memset(i8* %new53, i32 0, i32 %dim52)
%new54 = bitcast double* %horzcat_res49 to i8*
%new55 = getelementptr i8* %new54, i8 -16
%1 = call i32 @memcpy(i8* %new53, i8* %new55, i32 %dim52)
tail call void @free(i8* %new55)
%new56 = getelementptr i8* %new53, i8 16
%cp57 = bitcast i8* %new56 to double*
%new58 = alloca i8, i32 24
call void @memset(i8* %new58, i32 0, i32 24)
%new59 = getelementptr i8* %new58, i8 16
%dim60 = bitcast i8* %new59 to i32*
%dim61 = getelementptr i32* %dim60, i32 -1
store i32 24, i32* %dim61
%dim62 = bitcast i8* %new59 to i32*
%dim63 = getelementptr i32* %dim62, i32 -2
store i32 1, i32* %dim63
%dim64 = bitcast i8* %new59 to i32*
%dim65 = getelementptr i32* %dim64, i32 -4
store i32 0, i32* %dim65
%dim66 = bitcast i8* %new59 to i32*
%dim67 = getelementptr i32* %dim66, i32 -3
store i32 0, i32* %dim67
%new68 = bitcast i8* %new59 to i32*
%dim69 = getelementptr i32* %new68, i32 -4
store i32 1, i32* %dim69
%dim70 = getelementptr i32* %new68, i32 -3
```

```
store i32 1, i32* %dim70
%new71 = bitcast i32* %new68 to double*
%put72 = getelementptr double* %new71, i32 0
store double 5.000000e+00, double* %put72
%horzcat_res73 = call double* @horzcat(double* %new12, double* %new71)
%dim74 = bitcast double* %horzcat_res73 to i32*
%dim75 = getelementptr i32* %dim74, i32 -1
%dim76 = load i32* %dim75
%new77 = alloca i8, i32 %dim76
call void @memset(i8* %new77, i32 0, i32 %dim76)
%new78 = bitcast double* %horzcat_res73 to i8*
%new79 = getelementptr i8* %new78, i8 -16
%2 = call i32 @memcpy(i8* %new77, i8* %new79, i32 %dim76)
tail call void @free(i8* %new79)
%new80 = getelementptr i8* %new77, i8 16
%cp81 = bitcast i8* %new80 to double*
%new82 = alloca i8, i32 24
call void @memset(i8* %new82, i32 0, i32 24)
%new83 = getelementptr i8* %new82, i8 16
%dim84 = bitcast i8* %new83 to i32*
%dim85 = getelementptr i32* %dim84, i32 -1
store i32 24, i32* %dim85
%dim86 = bitcast i8* %new83 to i32*
%dim87 = getelementptr i32* %dim86, i32 -2
store i32 1, i32* %dim87
%dim88 = bitcast i8* %new83 to i32*
%dim89 = getelementptr i32* %dim88, i32 -4
store i32 0, i32* %dim89
%dim90 = bitcast i8* %new83 to i32*
%dim91 = getelementptr i32* %dim90, i32 -3
store i32 0, i32* %dim91
%new92 = bitcast i8* %new83 to i32*
%dim93 = getelementptr i32* %new92, i32 -4
store i32 1, i32* %dim93
%dim94 = getelementptr i32* %new92, i32 -3
store i32 1, i32* %dim94
%new95 = bitcast i32* %new92 to double*
%put96 = getelementptr double* %new95, i32 0
store double 1.500000e+00, double* %put96
%horzcat_res97 = call double* @horzcat(double* %new12, double* %new95)
%dim98 = bitcast double* %horzcat_res97 to i32*
%dim99 = getelementptr i32* %dim98, i32 -1
%dim100 = load i32* %dim99
%new101 = alloca i8, i32 %dim100
call void @memset(i8* %new101, i32 0, i32 %dim100)
%new102 = bitcast double* %horzcat_res97 to i8*
%new103 = getelementptr i8* %new102, i8 -16
%3 = call i32 @memcpy(i8* %new101, i8* %new103, i32 %dim100)
tail call void @free(i8* %new103)
%new104 = getelementptr i8* %new101, i8 16
%cp105 = bitcast i8* %new104 to double*
%new106 = alloca i8, i32 24
call void @memset(i8* %new106, i32 0, i32 24)
%new107 = getelementptr i8* %new106, i8 16
%dim108 = bitcast i8* %new107 to i32*
%dim109 = getelementptr i32* %dim108, i32 -1
store i32 24, i32* %dim109
%dim110 = bitcast i8* %new107 to i32*
%dim111 = getelementptr i32* %dim110, i32 -2
store i32 1, i32* %dim111
%dim112 = bitcast i8* %new107 to i32*
```

```
  %dim113 = getelementptr i32* %dim112, i32 -4
  store i32 0, i32* %dim113
  %dim114 = bitcast i8* %new107 to i32*
  %dim115 = getelementptr i32* %dim114, i32 -3
  store i32 0, i32* %dim115
  %new116 = bitcast i8* %new107 to i32*
  %dim117 = getelementptr i32* %new116, i32 -4
  store i32 1, i32* %dim117
  %dim118 = getelementptr i32* %new116, i32 -3
  store i32 1, i32* %dim118
  %new119 = bitcast i32* %new116 to double*
  %put120 = getelementptr double* %new119, i32 0
  store double 5.000000e-01, double* %put120
  %horzcat_res121 = call double* @horzcat(double* %new12, double* %new119)
  %dim122 = bitcast double* %horzcat_res121 to i32*
  %dim123 = getelementptr i32* %dim122, i32 -1
  %dim124 = load i32* %dim123
  %new125 = alloca i8, i32 %dim124
  call void @memset(i8* %new125, i32 0, i32 %dim124)
  %new126 = bitcast double* %horzcat_res121 to i8*
  %new127 = getelementptr i8* %new126, i8 -16
  %4 = call i32 @memcpy(i8* %new125, i8* %new127, i32 %dim124)
  tail call void @free(i8* %new127)
  %new128 = getelementptr i8* %new125, i8 16
  %cp129 = bitcast i8* %new128 to double*
  %new130 = alloca i8, i32 24
  call void @memset(i8* %new130, i32 0, i32 24)
  %new131 = getelementptr i8* %new130, i8 16
  %dim132 = bitcast i8* %new131 to i32*
  %dim133 = getelementptr i32* %dim132, i32 -1
  store i32 24, i32* %dim133
  %dim134 = bitcast i8* %new131 to i32*
  %dim135 = getelementptr i32* %dim134, i32 -2
  store i32 1, i32* %dim135
  %dim136 = bitcast i8* %new131 to i32*
  %dim137 = getelementptr i32* %dim136, i32 -4
  store i32 0, i32* %dim137
  %dim138 = bitcast i8* %new131 to i32*
  %dim139 = getelementptr i32* %dim138, i32 -3
  store i32 0, i32* %dim139
  %new140 = bitcast i8* %new131 to i32*
  %dim141 = getelementptr i32* %new140, i32 -4
  store i32 1, i32* %dim141
  %dim142 = getelementptr i32* %new140, i32 -3
  store i32 1, i32* %dim142
  %new143 = bitcast i32* %new140 to double*
  %put144 = getelementptr double* %new143, i32 0
  store double 2.000000e+00, double* %put144
  %horzcat_res145 = call double* @horzcat(double* %new12, double* %new143)
  %dim146 = bitcast double* %horzcat_res145 to i32*
  %dim147 = getelementptr i32* %dim146, i32 -1
  %dim148 = load i32* %dim147
  %new149 = alloca i8, i32 %dim148
  call void @memset(i8* %new149, i32 0, i32 %dim148)
  %new150 = bitcast double* %horzcat_res145 to i8*
  %new151 = getelementptr i8* %new150, i8 -16
  %5 = call i32 @memcpy(i8* %new149, i8* %new151, i32 %dim148)
  tail call void @free(i8* %new151)
  %new152 = getelementptr i8* %new149, i8 16
  %cp153 = bitcast i8* %new152 to double*
  %vertcat_res = call double* @vertcat(double* %new12, double* %cp153)
```

```
%dim154 = bitcast double* %vertcat_res to i32*
%dim155 = getelementptr i32* %dim154, i32 -1
%dim156 = load i32* %dim155
%new157 = alloca i8, i32 %dim156
call void @memset(i8* %new157, i32 0, i32 %dim156)
%new158 = bitcast double* %vertcat_res to i8*
%new159 = getelementptr i8* %new158, i8 -16
%6 = call i32 @memcpy(i8* %new157, i8* %new159, i32 %dim156)
tail call void @free(i8* %new159)
%new160 = getelementptr i8* %new157, i8 16
%cp161 = bitcast i8* %new160 to double*
%vertcat_res162 = call double* @vertcat(double* %cp161, double* %cp129)
%dim163 = bitcast double* %vertcat_res162 to i32*
%dim164 = getelementptr i32* %dim163, i32 -1
%dim165 = load i32* %dim164
%new166 = alloca i8, i32 %dim165
call void @memset(i8* %new166, i32 0, i32 %dim165)
%new167 = bitcast double* %vertcat_res162 to i8*
%new168 = getelementptr i8* %new167, i8 -16
%7 = call i32 @memcpy(i8* %new166, i8* %new168, i32 %dim165)
tail call void @free(i8* %new168)
%new169 = getelementptr i8* %new166, i8 16
%cp170 = bitcast i8* %new169 to double*
%vertcat_res171 = call double* @vertcat(double* %cp170, double* %cp105)
%dim172 = bitcast double* %vertcat_res171 to i32*
%dim173 = getelementptr i32* %dim172, i32 -1
%dim174 = load i32* %dim173
%new175 = alloca i8, i32 %dim174
call void @memset(i8* %new175, i32 0, i32 %dim174)
%new176 = bitcast double* %vertcat_res171 to i8*
%new177 = getelementptr i8* %new176, i8 -16
%8 = call i32 @memcpy(i8* %new175, i8* %new177, i32 %dim174)
tail call void @free(i8* %new177)
%new178 = getelementptr i8* %new175, i8 16
%cp179 = bitcast i8* %new178 to double*
%vertcat_res180 = call double* @vertcat(double* %cp179, double* %cp81)
%dim181 = bitcast double* %vertcat_res180 to i32*
%dim182 = getelementptr i32* %dim181, i32 -1
%dim183 = load i32* %dim182
%new184 = alloca i8, i32 %dim183
call void @memset(i8* %new184, i32 0, i32 %dim183)
%new185 = bitcast double* %vertcat_res180 to i8*
%new186 = getelementptr i8* %new185, i8 -16
%9 = call i32 @memcpy(i8* %new184, i8* %new186, i32 %dim183)
tail call void @free(i8* %new186)
%new187 = getelementptr i8* %new184, i8 16
%cp188 = bitcast i8* %new187 to double*
%vertcat_res189 = call double* @vertcat(double* %cp188, double* %cp57)
%dim190 = bitcast double* %vertcat_res189 to i32*
%dim191 = getelementptr i32* %dim190, i32 -1
%dim192 = load i32* %dim191
%new193 = alloca i8, i32 %dim192
call void @memset(i8* %new193, i32 0, i32 %dim192)
%new194 = bitcast double* %vertcat_res189 to i8*
%new195 = getelementptr i8* %new194, i8 -16
%10 = call i32 @memcpy(i8* %new193, i8* %new195, i32 %dim192)
tail call void @free(i8* %new195)
%new196 = getelementptr i8* %new193, i8 16
%cp197 = bitcast i8* %new196 to double*
%vertcat_res198 = call double* @vertcat(double* %cp197, double* %cp)
%dim199 = bitcast double* %vertcat_res198 to i32*
```

```
  %dim200 = getelementptr i32* %dim199, i32 -1
  %dim201 = load i32* %dim200
  %new202 = alloca i8, i32 %dim201
  call void @memset(i8* %new202, i32 0, i32 %dim201)
  %new203 = bitcast double* %vertcat_res198 to i8*
  %new204 = getelementptr i8* %new203, i8 -16
  %11 = call i32 @memcpy(i8* %new202, i8* %new204, i32 %dim201)
  tail call void @free(i8* %new204)
  %new205 = getelementptr i8* %new202, i8 16
  %cp206 = bitcast i8* %new205 to double*
  store double* %cp206, double** %y
  %new207 = alloca i8, i32 16
  call void @memset(i8* %new207, i32 0, i32 16)
  %new208 = getelementptr i8* %new207, i8 16
  %dim209 = bitcast i8* %new208 to i32*
  %dim210 = getelementptr i32* %dim209, i32 -1
  store i32 16, i32* %dim210
  %dim211 = bitcast i8* %new208 to i32*
  %dim212 = getelementptr i32* %dim211, i32 -2
  store i32 0, i32* %dim212
  %dim213 = bitcast i8* %new208 to i32*
  %dim214 = getelementptr i32* %dim213, i32 -4
  store i32 0, i32* %dim214
  %dim215 = bitcast i8* %new208 to i32*
  %dim216 = getelementptr i32* %dim215, i32 -3
  store i32 0, i32* %dim216
  %new217 = bitcast i8* %new208 to i32*
  %dim218 = getelementptr i32* %new217, i32 -4
  store i32 0, i32* %dim218
  %dim219 = getelementptr i32* %new217, i32 -3
  store i32 0, i32* %dim219
  %new220 = bitcast i32* %new217 to double*
  %new221 = alloca i8, i32 24
  call void @memset(i8* %new221, i32 0, i32 24)
  %new222 = getelementptr i8* %new221, i8 16
  %dim223 = bitcast i8* %new222 to i32*
  %dim224 = getelementptr i32* %dim223, i32 -1
  store i32 24, i32* %dim224
  %dim225 = bitcast i8* %new222 to i32*
  %dim226 = getelementptr i32* %dim225, i32 -2
  store i32 1, i32* %dim226
  %dim227 = bitcast i8* %new222 to i32*
  %dim228 = getelementptr i32* %dim227, i32 -4
  store i32 0, i32* %dim228
  %dim229 = bitcast i8* %new222 to i32*
  %dim230 = getelementptr i32* %dim229, i32 -3
  store i32 0, i32* %dim230
  %new231 = bitcast i8* %new222 to i32*
  %dim232 = getelementptr i32* %new231, i32 -4
  store i32 1, i32* %dim232
  %dim233 = getelementptr i32* %new231, i32 -3
  store i32 1, i32* %dim233
  %new234 = bitcast i32* %new231 to double*
  %put235 = getelementptr double* %new234, i32 0
  store double 6.000000e+00, double* %put235
  %horzcat_res236 = call double* @horzcat(double* %new220, double* %new234)
  %dim237 = bitcast double* %horzcat_res236 to i32*
  %dim238 = getelementptr i32* %dim237, i32 -1
  %dim239 = load i32* %dim238
  %new240 = alloca i8, i32 %dim239
  call void @memset(i8* %new240, i32 0, i32 %dim239)
```

```
%new241 = bitcast double* %horzcat_res236 to i8*
%new242 = getelementptr i8* %new241, i8 -16
%12 = call i32 @memcpy(i8* %new240, i8* %new242, i32 %dim239)
tail call void @free(i8* %new242)
%new243 = getelementptr i8* %new240, i8 16
%cp244 = bitcast i8* %new243 to double*
%new245 = alloca i8, i32 24
call void @memset(i8* %new245, i32 0, i32 24)
%new246 = getelementptr i8* %new245, i8 16
%dim247 = bitcast i8* %new246 to i32*
%dim248 = getelementptr i32* %dim247, i32 -1
store i32 24, i32* %dim248
%dim249 = bitcast i8* %new246 to i32*
%dim250 = getelementptr i32* %dim249, i32 -2
store i32 1, i32* %dim250
%dim251 = bitcast i8* %new246 to i32*
%dim252 = getelementptr i32* %dim251, i32 -4
store i32 0, i32* %dim252
%dim253 = bitcast i8* %new246 to i32*
%dim254 = getelementptr i32* %dim253, i32 -3
store i32 0, i32* %dim254
%new255 = bitcast i8* %new246 to i32*
%dim256 = getelementptr i32* %new255, i32 -4
store i32 1, i32* %dim256
%dim257 = getelementptr i32* %new255, i32 -3
store i32 1, i32* %dim257
%new258 = bitcast i32* %new255 to double*
%put259 = getelementptr double* %new258, i32 0
store double 6.000000e+00, double* %put259
%horzcat_res260 = call double* @horzcat(double* %cp244, double* %new258)
%dim261 = bitcast double* %horzcat_res260 to i32*
%dim262 = getelementptr i32* %dim261, i32 -1
%dim263 = load i32* %dim262
%new264 = alloca i8, i32 %dim263
call void @memset(i8* %new264, i32 0, i32 %dim263)
%new265 = bitcast double* %horzcat_res260 to i8*
%new266 = getelementptr i8* %new265, i8 -16
%13 = call i32 @memcpy(i8* %new264, i8* %new266, i32 %dim263)
tail call void @free(i8* %new266)
%new267 = getelementptr i8* %new264, i8 16
%cp268 = bitcast i8* %new267 to double*
%new269 = alloca i8, i32 24
call void @memset(i8* %new269, i32 0, i32 24)
%new270 = getelementptr i8* %new269, i8 16
%dim271 = bitcast i8* %new270 to i32*
%dim272 = getelementptr i32* %dim271, i32 -1
store i32 24, i32* %dim272
%dim273 = bitcast i8* %new270 to i32*
%dim274 = getelementptr i32* %dim273, i32 -2
store i32 1, i32* %dim274
%dim275 = bitcast i8* %new270 to i32*
%dim276 = getelementptr i32* %dim275, i32 -4
store i32 0, i32* %dim276
%dim277 = bitcast i8* %new270 to i32*
%dim278 = getelementptr i32* %dim277, i32 -3
store i32 0, i32* %dim278
%new279 = bitcast i8* %new270 to i32*
%dim280 = getelementptr i32* %new279, i32 -4
store i32 1, i32* %dim280
%dim281 = getelementptr i32* %new279, i32 -3
store i32 1, i32* %dim281
```

```
%new282 = bitcast i32* %new279 to double*
%put283 = getelementptr double* %new282, i32 0
store double 5.000000e+00, double* %put283
%horzcat_res284 = call double* @horzcat(double* %new220, double* %new282)
%dim285 = bitcast double* %horzcat_res284 to i32*
%dim286 = getelementptr i32* %dim285, i32 -1
%dim287 = load i32* %dim286
%new288 = alloca i8, i32 %dim287
call void @memset(i8* %new288, i32 0, i32 %dim287)
%new289 = bitcast double* %horzcat_res284 to i8*
%new290 = getelementptr i8* %new289, i8 -16
%14 = call i32 @memcpy(i8* %new288, i8* %new290, i32 %dim287)
tail call void @free(i8* %new290)
%new291 = getelementptr i8* %new288, i8 16
%cp292 = bitcast i8* %new291 to double*
%new293 = alloca i8, i32 24
call void @memset(i8* %new293, i32 0, i32 24)
%new294 = getelementptr i8* %new293, i8 16
%dim295 = bitcast i8* %new294 to i32*
%dim296 = getelementptr i32* %dim295, i32 -1
store i32 24, i32* %dim296
%dim297 = bitcast i8* %new294 to i32*
%dim298 = getelementptr i32* %dim297, i32 -2
store i32 1, i32* %dim298
%dim299 = bitcast i8* %new294 to i32*
%dim300 = getelementptr i32* %dim299, i32 -4
store i32 0, i32* %dim300
%dim301 = bitcast i8* %new294 to i32*
%dim302 = getelementptr i32* %dim301, i32 -3
store i32 0, i32* %dim302
%new303 = bitcast i8* %new294 to i32*
%dim304 = getelementptr i32* %new303, i32 -4
store i32 1, i32* %dim304
%dim305 = getelementptr i32* %new303, i32 -3
store i32 1, i32* %dim305
%new306 = bitcast i32* %new303 to double*
%put307 = getelementptr double* %new306, i32 0
store double 5.000000e+00, double* %put307
%horzcat_res308 = call double* @horzcat(double* %cp292, double* %new306)
%dim309 = bitcast double* %horzcat_res308 to i32*
%dim310 = getelementptr i32* %dim309, i32 -1
%dim311 = load i32* %dim310
%new312 = alloca i8, i32 %dim311
call void @memset(i8* %new312, i32 0, i32 %dim311)
%new313 = bitcast double* %horzcat_res308 to i8*
%new314 = getelementptr i8* %new313, i8 -16
%15 = call i32 @memcpy(i8* %new312, i8* %new314, i32 %dim311)
tail call void @free(i8* %new314)
%new315 = getelementptr i8* %new312, i8 16
%cp316 = bitcast i8* %new315 to double*
%new317 = alloca i8, i32 24
call void @memset(i8* %new317, i32 0, i32 24)
%new318 = getelementptr i8* %new317, i8 16
%dim319 = bitcast i8* %new318 to i32*
%dim320 = getelementptr i32* %dim319, i32 -1
store i32 24, i32* %dim320
%dim321 = bitcast i8* %new318 to i32*
%dim322 = getelementptr i32* %dim321, i32 -2
store i32 1, i32* %dim322
%dim323 = bitcast i8* %new318 to i32*
%dim324 = getelementptr i32* %dim323, i32 -4
```

```
store i32 0, i32* %dim324
%dim325 = bitcast i8* %new318 to i32*
%dim326 = getelementptr i32* %dim325, i32 -3
store i32 0, i32* %dim326
%new327 = bitcast i8* %new318 to i32*
%dim328 = getelementptr i32* %new327, i32 -4
store i32 1, i32* %dim328
%dim329 = getelementptr i32* %new327, i32 -3
store i32 1, i32* %dim329
%new330 = bitcast i32* %new327 to double*
%put331 = getelementptr double* %new330, i32 0
store double 4.000000e+00, double* %put331
%horzcat_res332 = call double* @horzcat(double* %new220, double* %new330)
%dim333 = bitcast double* %horzcat_res332 to i32*
%dim334 = getelementptr i32* %dim333, i32 -1
%dim335 = load i32* %dim334
%new336 = alloca i8, i32 %dim335
call void @memset(i8* %new336, i32 0, i32 %dim335)
%new337 = bitcast double* %horzcat_res332 to i8*
%new338 = getelementptr i8* %new337, i8 -16
%16 = call i32 @memcpy(i8* %new336, i8* %new338, i32 %dim335)
tail call void @free(i8* %new338)
%new339 = getelementptr i8* %new336, i8 16
%cp340 = bitcast i8* %new339 to double*
%new341 = alloca i8, i32 24
call void @memset(i8* %new341, i32 0, i32 24)
%new342 = getelementptr i8* %new341, i8 16
%dim343 = bitcast i8* %new342 to i32*
%dim344 = getelementptr i32* %dim343, i32 -1
store i32 24, i32* %dim344
%dim345 = bitcast i8* %new342 to i32*
%dim346 = getelementptr i32* %dim345, i32 -2
store i32 1, i32* %dim346
%dim347 = bitcast i8* %new342 to i32*
%dim348 = getelementptr i32* %dim347, i32 -4
store i32 0, i32* %dim348
%dim349 = bitcast i8* %new342 to i32*
%dim350 = getelementptr i32* %dim349, i32 -3
store i32 0, i32* %dim350
%new351 = bitcast i8* %new342 to i32*
%dim352 = getelementptr i32* %new351, i32 -4
store i32 1, i32* %dim352
%dim353 = getelementptr i32* %new351, i32 -3
store i32 1, i32* %dim353
%new354 = bitcast i32* %new351 to double*
%put355 = getelementptr double* %new354, i32 0
store double 3.000000e+00, double* %put355
%horzcat_res356 = call double* @horzcat(double* %cp340, double* %new354)
%dim357 = bitcast double* %horzcat_res356 to i32*
%dim358 = getelementptr i32* %dim357, i32 -1
%dim359 = load i32* %dim358
%new360 = alloca i8, i32 %dim359
call void @memset(i8* %new360, i32 0, i32 %dim359)
%new361 = bitcast double* %horzcat_res356 to i8*
%new362 = getelementptr i8* %new361, i8 -16
%17 = call i32 @memcpy(i8* %new360, i8* %new362, i32 %dim359)
tail call void @free(i8* %new362)
%new363 = getelementptr i8* %new360, i8 16
%cp364 = bitcast i8* %new363 to double*
%new365 = alloca i8, i32 24
call void @memset(i8* %new365, i32 0, i32 24)
```

```
  %new366 = getelementptr i8* %new365, i8 16
  %dim367 = bitcast i8* %new366 to i32*
  %dim368 = getelementptr i32* %dim367, i32 -1
  store i32 24, i32* %dim368
  %dim369 = bitcast i8* %new366 to i32*
  %dim370 = getelementptr i32* %dim369, i32 -2
  store i32 1, i32* %dim370
  %dim371 = bitcast i8* %new366 to i32*
  %dim372 = getelementptr i32* %dim371, i32 -4
  store i32 0, i32* %dim372
  %dim373 = bitcast i8* %new366 to i32*
  %dim374 = getelementptr i32* %dim373, i32 -3
  store i32 0, i32* %dim374
  %new375 = bitcast i8* %new366 to i32*
  %dim376 = getelementptr i32* %new375, i32 -4
  store i32 1, i32* %dim376
  %dim377 = getelementptr i32* %new375, i32 -3
  store i32 1, i32* %dim377
  %new378 = bitcast i32* %new375 to double*
  %put379 = getelementptr double* %new378, i32 0
  store double 3.000000e+00, double* %put379
  %horzcat_res380 = call double* @horzcat(double* %new220, double* %new378)
  %dim381 = bitcast double* %horzcat_res380 to i32*
  %dim382 = getelementptr i32* %dim381, i32 -1
  %dim383 = load i32* %dim382
  %new384 = alloca i8, i32 %dim383
  call void @memset(i8* %new384, i32 0, i32 %dim383)
  %new385 = bitcast double* %horzcat_res380 to i8*
  %new386 = getelementptr i8* %new385, i8 -16
  %18 = call i32 @memcpy(i8* %new384, i8* %new386, i32 %dim383)
  tail call void @free(i8* %new386)
  %new387 = getelementptr i8* %new384, i8 16
  %cp388 = bitcast i8* %new387 to double*
  %new389 = alloca i8, i32 24
  call void @memset(i8* %new389, i32 0, i32 24)
  %new390 = getelementptr i8* %new389, i8 16
  %dim391 = bitcast i8* %new390 to i32*
  %dim392 = getelementptr i32* %dim391, i32 -1
  store i32 24, i32* %dim392
  %dim393 = bitcast i8* %new390 to i32*
  %dim394 = getelementptr i32* %dim393, i32 -2
  store i32 1, i32* %dim394
  %dim395 = bitcast i8* %new390 to i32*
  %dim396 = getelementptr i32* %dim395, i32 -4
  store i32 0, i32* %dim396
  %dim397 = bitcast i8* %new390 to i32*
  %dim398 = getelementptr i32* %dim397, i32 -3
  store i32 0, i32* %dim398
  %new399 = bitcast i8* %new390 to i32*
  %dim400 = getelementptr i32* %new399, i32 -4
  store i32 1, i32* %dim400
  %dim401 = getelementptr i32* %new399, i32 -3
  store i32 1, i32* %dim401
  %new402 = bitcast i32* %new399 to double*
  %put403 = getelementptr double* %new402, i32 0
  store double 3.000000e+00, double* %put403
  %horzcat_res404 = call double* @horzcat(double* %cp388, double* %new402)
  %dim405 = bitcast double* %horzcat_res404 to i32*
  %dim406 = getelementptr i32* %dim405, i32 -1
  %dim407 = load i32* %dim406
  %new408 = alloca i8, i32 %dim407
```

```
call void @memset(i8* %new408, i32 0, i32 %dim407)
%new409 = bitcast double* %horzcat_res404 to i8*
%new410 = getelementptr i8* %new409, i8 -16
%19 = call i32 @memcpy(i8* %new408, i8* %new410, i32 %dim407)
tail call void @free(i8* %new410)
%new411 = getelementptr i8* %new408, i8 16
%cp412 = bitcast i8* %new411 to double*
%new413 = alloca i8, i32 24
call void @memset(i8* %new413, i32 0, i32 24)
%new414 = getelementptr i8* %new413, i8 16
%dim415 = bitcast i8* %new414 to i32*
%dim416 = getelementptr i32* %dim415, i32 -1
store i32 24, i32* %dim416
%dim417 = bitcast i8* %new414 to i32*
%dim418 = getelementptr i32* %dim417, i32 -2
store i32 1, i32* %dim418
%dim419 = bitcast i8* %new414 to i32*
%dim420 = getelementptr i32* %dim419, i32 -4
store i32 0, i32* %dim420
%dim421 = bitcast i8* %new414 to i32*
%dim422 = getelementptr i32* %dim421, i32 -3
store i32 0, i32* %dim422
%new423 = bitcast i8* %new414 to i32*
%dim424 = getelementptr i32* %new423, i32 -4
store i32 1, i32* %dim424
%dim425 = getelementptr i32* %new423, i32 -3
store i32 1, i32* %dim425
%new426 = bitcast i32* %new423 to double*
%put427 = getelementptr double* %new426, i32 0
store double 2.000000e+00, double* %put427
%horzcat_res428 = call double* @horzcat(double* %new220, double* %new426)
%dim429 = bitcast double* %horzcat_res428 to i32*
%dim430 = getelementptr i32* %dim429, i32 -1
%dim431 = load i32* %dim430
%new432 = alloca i8, i32 %dim431
call void @memset(i8* %new432, i32 0, i32 %dim431)
%new433 = bitcast double* %horzcat_res428 to i8*
%new434 = getelementptr i8* %new433, i8 -16
%20 = call i32 @memcpy(i8* %new432, i8* %new434, i32 %dim431)
tail call void @free(i8* %new434)
%new435 = getelementptr i8* %new432, i8 16
%cp436 = bitcast i8* %new435 to double*
%new437 = alloca i8, i32 24
call void @memset(i8* %new437, i32 0, i32 24)
%new438 = getelementptr i8* %new437, i8 16
%dim439 = bitcast i8* %new438 to i32*
%dim440 = getelementptr i32* %dim439, i32 -1
store i32 24, i32* %dim440
%dim441 = bitcast i8* %new438 to i32*
%dim442 = getelementptr i32* %dim441, i32 -2
store i32 1, i32* %dim442
%dim443 = bitcast i8* %new438 to i32*
%dim444 = getelementptr i32* %dim443, i32 -4
store i32 0, i32* %dim444
%dim445 = bitcast i8* %new438 to i32*
%dim446 = getelementptr i32* %dim445, i32 -3
store i32 0, i32* %dim446
%new447 = bitcast i8* %new438 to i32*
%dim448 = getelementptr i32* %new447, i32 -4
store i32 1, i32* %dim448
%dim449 = getelementptr i32* %new447, i32 -3
```

```
store i32 1, i32* %dim449
%new450 = bitcast i32* %new447 to double*
%put451 = getelementptr double* %new450, i32 0
store double 2.000000e+00, double* %put451
%horzcat_res452 = call double* @horzcat(double* %cp436, double* %new450)
%dim453 = bitcast double* %horzcat_res452 to i32*
%dim454 = getelementptr i32* %dim453, i32 -1
%dim455 = load i32* %dim454
%new456 = alloca i8, i32 %dim455
call void @memset(i8* %new456, i32 0, i32 %dim455)
%new457 = bitcast double* %horzcat_res452 to i8*
%new458 = getelementptr i8* %new457, i8 -16
%21 = call i32 @memcpy(i8* %new456, i8* %new458, i32 %dim455)
tail call void @free(i8* %new458)
%new459 = getelementptr i8* %new456, i8 16
%cp460 = bitcast i8* %new459 to double*
%new461 = alloca i8, i32 24
call void @memset(i8* %new461, i32 0, i32 24)
%new462 = getelementptr i8* %new461, i8 16
%dim463 = bitcast i8* %new462 to i32*
%dim464 = getelementptr i32* %dim463, i32 -1
store i32 24, i32* %dim464
%dim465 = bitcast i8* %new462 to i32*
%dim466 = getelementptr i32* %dim465, i32 -2
store i32 1, i32* %dim466
%dim467 = bitcast i8* %new462 to i32*
%dim468 = getelementptr i32* %dim467, i32 -4
store i32 0, i32* %dim468
%dim469 = bitcast i8* %new462 to i32*
%dim470 = getelementptr i32* %dim469, i32 -3
store i32 0, i32* %dim470
%new471 = bitcast i8* %new462 to i32*
%dim472 = getelementptr i32* %new471, i32 -4
store i32 1, i32* %dim472
%dim473 = getelementptr i32* %new471, i32 -3
store i32 1, i32* %dim473
%new474 = bitcast i32* %new471 to double*
%put475 = getelementptr double* %new474, i32 0
store double 1.000000e+00, double* %put475
%horzcat_res476 = call double* @horzcat(double* %new220, double* %new474)
%dim477 = bitcast double* %horzcat_res476 to i32*
%dim478 = getelementptr i32* %dim477, i32 -1
%dim479 = load i32* %dim478
%new480 = alloca i8, i32 %dim479
call void @memset(i8* %new480, i32 0, i32 %dim479)
%new481 = bitcast double* %horzcat_res476 to i8*
%new482 = getelementptr i8* %new481, i8 -16
%22 = call i32 @memcpy(i8* %new480, i8* %new482, i32 %dim479)
tail call void @free(i8* %new482)
%new483 = getelementptr i8* %new480, i8 16
%cp484 = bitcast i8* %new483 to double*
%new485 = alloca i8, i32 24
call void @memset(i8* %new485, i32 0, i32 24)
%new486 = getelementptr i8* %new485, i8 16
%dim487 = bitcast i8* %new486 to i32*
%dim488 = getelementptr i32* %dim487, i32 -1
store i32 24, i32* %dim488
%dim489 = bitcast i8* %new486 to i32*
%dim490 = getelementptr i32* %dim489, i32 -2
store i32 1, i32* %dim490
%dim491 = bitcast i8* %new486 to i32*
```

```
%dim492 = getelementptr i32* %dim491, i32 -4
store i32 0, i32* %dim492
%dim493 = bitcast i8* %new486 to i32*
%dim494 = getelementptr i32* %dim493, i32 -3
store i32 0, i32* %dim494
%new495 = bitcast i8* %new486 to i32*
%dim496 = getelementptr i32* %new495, i32 -4
store i32 1, i32* %dim496
%dim497 = getelementptr i32* %new495, i32 -3
store i32 1, i32* %dim497
%new498 = bitcast i32* %new495 to double*
%put499 = getelementptr double* %new498, i32 0
store double 2.000000e+00, double* %put499
%horzcat_res500 = call double* @horzcat(double* %cp484, double* %new498)
%dim501 = bitcast double* %horzcat_res500 to i32*
%dim502 = getelementptr i32* %dim501, i32 -1
%dim503 = load i32* %dim502
%new504 = alloca i8, i32 %dim503
call void @memset(i8* %new504, i32 0, i32 %dim503)
%new505 = bitcast double* %horzcat_res500 to i8*
%new506 = getelementptr i8* %new505, i8 -16
%23 = call i32 @memcpy(i8* %new504, i8* %new506, i32 %dim503)
tail call void @free(i8* %new506)
%new507 = getelementptr i8* %new504, i8 16
%cp508 = bitcast i8* %new507 to double*
%vertcat_res509 = call double* @vertcat(double* %new220, double* %cp508)
%dim510 = bitcast double* %vertcat_res509 to i32*
%dim511 = getelementptr i32* %dim510, i32 -1
%dim512 = load i32* %dim511
%new513 = alloca i8, i32 %dim512
call void @memset(i8* %new513, i32 0, i32 %dim512)
%new514 = bitcast double* %vertcat_res509 to i8*
%new515 = getelementptr i8* %new514, i8 -16
%24 = call i32 @memcpy(i8* %new513, i8* %new515, i32 %dim512)
tail call void @free(i8* %new515)
%new516 = getelementptr i8* %new513, i8 16
%cp517 = bitcast i8* %new516 to double*
%vertcat_res518 = call double* @vertcat(double* %cp517, double* %cp460)
%dim519 = bitcast double* %vertcat_res518 to i32*
%dim520 = getelementptr i32* %dim519, i32 -1
%dim521 = load i32* %dim520
%new522 = alloca i8, i32 %dim521
call void @memset(i8* %new522, i32 0, i32 %dim521)
%new523 = bitcast double* %vertcat_res518 to i8*
%new524 = getelementptr i8* %new523, i8 -16
%25 = call i32 @memcpy(i8* %new522, i8* %new524, i32 %dim521)
tail call void @free(i8* %new524)
%new525 = getelementptr i8* %new522, i8 16
%cp526 = bitcast i8* %new525 to double*
%vertcat_res527 = call double* @vertcat(double* %cp526, double* %cp412)
%dim528 = bitcast double* %vertcat_res527 to i32*
%dim529 = getelementptr i32* %dim528, i32 -1
%dim530 = load i32* %dim529
%new531 = alloca i8, i32 %dim530
call void @memset(i8* %new531, i32 0, i32 %dim530)
%new532 = bitcast double* %vertcat_res527 to i8*
%new533 = getelementptr i8* %new532, i8 -16
%26 = call i32 @memcpy(i8* %new531, i8* %new533, i32 %dim530)
tail call void @free(i8* %new533)
%new534 = getelementptr i8* %new531, i8 16
%cp535 = bitcast i8* %new534 to double*
```

```
%vertcat_res536 = call double* @vertcat(double* %cp535, double* %cp364)
%dim537 = bitcast double* %vertcat_res536 to i32*
%dim538 = getelementptr i32* %dim537, i32 -1
%dim539 = load i32* %dim538
%new540 = alloca i8, i32 %dim539
call void @memset(i8* %new540, i32 0, i32 %dim539)
%new541 = bitcast double* %vertcat_res536 to i8*
%new542 = getelementptr i8* %new541, i8 -16
%27 = call i32 @memcpy(i8* %new540, i8* %new542, i32 %dim539)
tail call void @free(i8* %new542)
%new543 = getelementptr i8* %new540, i8 16
%cp544 = bitcast i8* %new543 to double*
%vertcat_res545 = call double* @vertcat(double* %cp544, double* %cp316)
%dim546 = bitcast double* %vertcat_res545 to i32*
%dim547 = getelementptr i32* %dim546, i32 -1
%dim548 = load i32* %dim547
%new549 = alloca i8, i32 %dim548
call void @memset(i8* %new549, i32 0, i32 %dim548)
%new550 = bitcast double* %vertcat_res545 to i8*
%new551 = getelementptr i8* %new550, i8 -16
%28 = call i32 @memcpy(i8* %new549, i8* %new551, i32 %dim548)
tail call void @free(i8* %new551)
%new552 = getelementptr i8* %new549, i8 16
%cp553 = bitcast i8* %new552 to double*
%vertcat_res554 = call double* @vertcat(double* %cp553, double* %cp268)
%dim555 = bitcast double* %vertcat_res554 to i32*
%dim556 = getelementptr i32* %dim555, i32 -1
%dim557 = load i32* %dim556
%new558 = alloca i8, i32 %dim557
call void @memset(i8* %new558, i32 0, i32 %dim557)
%new559 = bitcast double* %vertcat_res554 to i8*
%new560 = getelementptr i8* %new559, i8 -16
%29 = call i32 @memcpy(i8* %new558, i8* %new560, i32 %dim557)
tail call void @free(i8* %new560)
%new561 = getelementptr i8* %new558, i8 16
%cp562 = bitcast i8* %new561 to double*
store double* %cp562, double** %x
%x563 = load double** %x
%rows_res = call i32 @rows(double* %x563)
%new564 = mul i32 %rows_res, 1
%new565 = mul i32 %new564, 8
%new566 = add i32 %new565, 16
%new567 = alloca i8, i32 %new566
call void @memset(i8* %new567, i32 0, i32 %new566)
%new568 = getelementptr i8* %new567, i8 16
%dim569 = bitcast i8* %new568 to i32*
%dim570 = getelementptr i32* %dim569, i32 -1
store i32 %new566, i32* %dim570
%dim571 = bitcast i8* %new568 to i32*
%dim572 = getelementptr i32* %dim571, i32 -2
store i32 %new564, i32* %dim572
%dim573 = bitcast i8* %new568 to i32*
%dim574 = getelementptr i32* %dim573, i32 -4
store i32 0, i32* %dim574
%dim575 = bitcast i8* %new568 to i32*
%dim576 = getelementptr i32* %dim575, i32 -3
store i32 0, i32* %dim576
%new577 = bitcast i8* %new568 to i32*
%dim578 = getelementptr i32* %new577, i32 -4
store i32 %rows_res, i32* %dim578
%dim579 = getelementptr i32* %new577, i32 -3
```

```
store i32 1, i32* %dim579
%new580 = bitcast i32* %new577 to double*
%new581 = alloca i8, i32 16
call void @memset(i8* %new581, i32 0, i32 16)
%new582 = getelementptr i8* %new581, i8 16
%dim583 = bitcast i8* %new582 to i32*
%dim584 = getelementptr i32* %dim583, i32 -1
store i32 16, i32* %dim584
%dim585 = bitcast i8* %new582 to i32*
%dim586 = getelementptr i32* %dim585, i32 -2
store i32 0, i32* %dim586
%dim587 = bitcast i8* %new582 to i32*
%dim588 = getelementptr i32* %dim587, i32 -4
store i32 0, i32* %dim588
%dim589 = bitcast i8* %new582 to i32*
%dim590 = getelementptr i32* %dim589, i32 -3
store i32 0, i32* %dim590
%new591 = bitcast i8* %new582 to i32*
%dim592 = getelementptr i32* %new591, i32 -4
store i32 0, i32* %dim592
%dim593 = getelementptr i32* %new591, i32 -3
store i32 0, i32* %dim593
%new594 = bitcast i32* %new591 to double*
%new595 = alloca i8, i32 24
call void @memset(i8* %new595, i32 0, i32 24)
%new596 = getelementptr i8* %new595, i8 16
%dim597 = bitcast i8* %new596 to i32*
%dim598 = getelementptr i32* %dim597, i32 -1
store i32 24, i32* %dim598
%dim599 = bitcast i8* %new596 to i32*
%dim600 = getelementptr i32* %dim599, i32 -2
store i32 1, i32* %dim600
%dim601 = bitcast i8* %new596 to i32*
%dim602 = getelementptr i32* %dim601, i32 -4
store i32 0, i32* %dim602
%dim603 = bitcast i8* %new596 to i32*
%dim604 = getelementptr i32* %dim603, i32 -3
store i32 0, i32* %dim604
%new605 = bitcast i8* %new596 to i32*
%dim606 = getelementptr i32* %new605, i32 -4
store i32 1, i32* %dim606
%dim607 = getelementptr i32* %new605, i32 -3
store i32 1, i32* %dim607
%new608 = bitcast i32* %new605 to double*
%put609 = getelementptr double* %new608, i32 0
store double 1.000000e+00, double* %put609
%horzcat_res610 = call double* @horzcat(double* %new594, double* %new608)
%dim611 = bitcast double* %horzcat_res610 to i32*
%dim612 = getelementptr i32* %dim611, i32 -1
%dim613 = load i32* %dim612
%new614 = alloca i8, i32 %dim613
call void @memset(i8* %new614, i32 0, i32 %dim613)
%new615 = bitcast double* %horzcat_res610 to i8*
%new616 = getelementptr i8* %new615, i8 -16
%30 = call i32 @memcpy(i8* %new614, i8* %new616, i32 %dim613)
tail call void @free(i8* %new616)
%new617 = getelementptr i8* %new614, i8 16
%cp618 = bitcast i8* %new617 to double*
%vertcat_res619 = call double* @vertcat(double* %new594, double* %cp618)
%dim620 = bitcast double* %vertcat_res619 to i32*
%dim621 = getelementptr i32* %dim620, i32 -1
```

```
%dim622 = load i32* %dim621
%new623 = alloca i8, i32 %dim622
call void @memset(i8* %new623, i32 0, i32 %dim622)
%new624 = bitcast double* %vertcat_res619 to i8*
%new625 = getelementptr i8* %new624, i8 -16
%31 = call i32 @memcpy(i8* %new623, i8* %new625, i32 %dim622)
tail call void @free(i8* %new625)
%new626 = getelementptr i8* %new623, i8 16
%cp627 = bitcast i8* %new626 to double*
%madd_res = call double* @madd(double* %new580, double* %cp627)
%dim628 = bitcast double* %madd_res to i32*
%dim629 = getelementptr i32* %dim628, i32 -1
%dim630 = load i32* %dim629
%new631 = alloca i8, i32 %dim630
call void @memset(i8* %new631, i32 0, i32 %dim630)
%new632 = bitcast double* %madd_res to i8*
%new633 = getelementptr i8* %new632, i8 -16
%32 = call i32 @memcpy(i8* %new631, i8* %new633, i32 %dim630)
tail call void @free(i8* %new633)
%new634 = getelementptr i8* %new631, i8 16
%cp635 = bitcast i8* %new634 to double*
store double* %cp635, double** %xx
%new636 = alloca i8, i32 16
call void @memset(i8* %new636, i32 0, i32 16)
%new637 = getelementptr i8* %new636, i8 16
%dim638 = bitcast i8* %new637 to i32*
%dim639 = getelementptr i32* %dim638, i32 -1
store i32 16, i32* %dim639
%dim640 = bitcast i8* %new637 to i32*
%dim641 = getelementptr i32* %dim640, i32 -2
store i32 0, i32* %dim641
%dim642 = bitcast i8* %new637 to i32*
%dim643 = getelementptr i32* %dim642, i32 -4
store i32 0, i32* %dim643
%dim644 = bitcast i8* %new637 to i32*
%dim645 = getelementptr i32* %dim644, i32 -3
store i32 0, i32* %dim645
%new646 = bitcast i8* %new637 to i32*
%dim647 = getelementptr i32* %new646, i32 -4
store i32 0, i32* %dim647
%dim648 = getelementptr i32* %new646, i32 -3
store i32 0, i32* %dim648
%new649 = bitcast i32* %new646 to double*
%x650 = load double** %x
%xx651 = load double** %xx
%horzcat_res652 = call double* @horzcat(double* %new649, double* %xx651)
%dim653 = bitcast double* %horzcat_res652 to i32*
%dim654 = getelementptr i32* %dim653, i32 -1
%dim655 = load i32* %dim654
%new656 = alloca i8, i32 %dim655
call void @memset(i8* %new656, i32 0, i32 %dim655)
%new657 = bitcast double* %horzcat_res652 to i8*
%new658 = getelementptr i8* %new657, i8 -16
%33 = call i32 @memcpy(i8* %new656, i8* %new658, i32 %dim655)
tail call void @free(i8* %new658)
%new659 = getelementptr i8* %new656, i8 16
%cp660 = bitcast i8* %new659 to double*
%horzcat_res661 = call double* @horzcat(double* %cp660, double* %x650)
%dim662 = bitcast double* %horzcat_res661 to i32*
%dim663 = getelementptr i32* %dim662, i32 -1
%dim664 = load i32* %dim663
```

```
  %new665 = alloca i8, i32 %dim664
  call void @memset(i8* %new665, i32 0, i32 %dim664)
  %new666 = bitcast double* %horzcat_res661 to i8*
  %new667 = getelementptr i8* %new666, i8 -16
  %34 = call i32 @memcpy(i8* %new665, i8* %new667, i32 %dim664)
  tail call void @free(i8* %new667)
  %new668 = getelementptr i8* %new665, i8 16
  %cp669 = bitcast i8* %new668 to double*
  %vertcat_res670 = call double* @vertcat(double* %new649, double* %cp669)
  %dim671 = bitcast double* %vertcat_res670 to i32*
  %dim672 = getelementptr i32* %dim671, i32 -1
  %dim673 = load i32* %dim672
  %new674 = alloca i8, i32 %dim673
  call void @memset(i8* %new674, i32 0, i32 %dim673)
  %new675 = bitcast double* %vertcat_res670 to i8*
  %new676 = getelementptr i8* %new675, i8 -16
  %35 = call i32 @memcpy(i8* %new674, i8* %new676, i32 %dim673)
  tail call void @free(i8* %new676)
  %new677 = getelementptr i8* %new674, i8 16
  %cp678 = bitcast i8* %new677 to double*
  store double* %cp678, double** %x
  %x679 = load double** %x
  %mtransp_res = call double* @mtransp(double* %x679)
  %dim680 = bitcast double* %mtransp_res to i32*
  %dim681 = getelementptr i32* %dim680, i32 -1
  %dim682 = load i32* %dim681
  %new683 = alloca i8, i32 %dim682
  call void @memset(i8* %new683, i32 0, i32 %dim682)
  %new684 = bitcast double* %mtransp_res to i8*
  %new685 = getelementptr i8* %new684, i8 -16
  %36 = call i32 @memcpy(i8* %new683, i8* %new685, i32 %dim682)
  tail call void @free(i8* %new685)
  %new686 = getelementptr i8* %new683, i8 16
  %cp687 = bitcast i8* %new686 to double*
  %x688 = load double** %x
  %mmul_res = call double* @mmul(double* %cp687, double* %x688)
  %dim689 = bitcast double* %mmul_res to i32*
  %dim690 = getelementptr i32* %dim689, i32 -1
  %dim691 = load i32* %dim690
  %new692 = alloca i8, i32 %dim691
  call void @memset(i8* %new692, i32 0, i32 %dim691)
  %new693 = bitcast double* %mmul_res to i8*
  %new694 = getelementptr i8* %new693, i8 -16
  %37 = call i32 @memcpy(i8* %new692, i8* %new694, i32 %dim691)
  tail call void @free(i8* %new694)
  %new695 = getelementptr i8* %new692, i8 16
  %cp696 = bitcast i8* %new695 to double*
  store double* %cp696, double** %xx
  %xx697 = load double** %xx
  %inv_res = call double* @inv(double* %xx697)
  %dim698 = bitcast double* %inv_res to i32*
  %dim699 = getelementptr i32* %dim698, i32 -1
  %dim700 = load i32* %dim699
  %new701 = alloca i8, i32 %dim700
  call void @memset(i8* %new701, i32 0, i32 %dim700)
  %new702 = bitcast double* %inv_res to i8*
  %new703 = getelementptr i8* %new702, i8 -16
  %38 = call i32 @memcpy(i8* %new701, i8* %new703, i32 %dim700)
  tail call void @free(i8* %new703)
  %new704 = getelementptr i8* %new701, i8 16
  %cp705 = bitcast i8* %new704 to double*
  call void @printmat(double* %cp705)
  %xx706 = load double** %xx
  %inv_res707 = call double* @inv(double* %xx706)
  %dim708 = bitcast double* %inv_res707 to i32*
  %dim709 = getelementptr i32* %dim708, i32 -1
  %dim710 = load i32* %dim709
  %new711 = alloca i8, i32 %dim710
  call void @memset(i8* %new711, i32 0, i32 %dim710)
  %new712 = bitcast double* %inv_res707 to i8*
  %new713 = getelementptr i8* %new712, i8 -16
  %39 = call i32 @memcpy(i8* %new711, i8* %new713, i32 %dim710)
  tail call void @free(i8* %new713)
  %new714 = getelementptr i8* %new711, i8 16
  %cp715 = bitcast i8* %new714 to double*
  %xx716 = load double** %xx
  %mmul_res717 = call double* @mmul(double* %cp715, double* %xx716)
  %dim718 = bitcast double* %mmul_res717 to i32*
  %dim719 = getelementptr i32* %dim718, i32 -1
  %dim720 = load i32* %dim719
  %new721 = alloca i8, i32 %dim720
  call void @memset(i8* %new721, i32 0, i32 %dim720)
  %new722 = bitcast double* %mmul_res717 to i8*
  %new723 = getelementptr i8* %new722, i8 -16
  %40 = call i32 @memcpy(i8* %new721, i8* %new723, i32 %dim720)
  tail call void @free(i8* %new723)
  %new724 = getelementptr i8* %new721, i8 16
  %cp725 = bitcast i8* %new724 to double*
  call void @printmat(double* %cp725)
  %y726 = load double** %y
  %x727 = load double** %x
  %y728 = load double** %y
  %x729 = load double** %x
  %mdiv_res = call double* @mdiv(double* %y728, double* %x729)
  %dim730 = bitcast double* %mdiv_res to i32*
  %dim731 = getelementptr i32* %dim730, i32 -1
  %dim732 = load i32* %dim731
  %new733 = alloca i8, i32 %dim732
  call void @memset(i8* %new733, i32 0, i32 %dim732)
  %new734 = bitcast double* %mdiv_res to i8*
  %new735 = getelementptr i8* %new734, i8 -16
  %41 = call i32 @memcpy(i8* %new733, i8* %new735, i32 %dim732)
  tail call void @free(i8* %new735)
  %new736 = getelementptr i8* %new733, i8 16
  %cp737 = bitcast i8* %new736 to double*
  %mmul_res738 = call double* @mmul(double* %x727, double* %cp737)
  %dim739 = bitcast double* %mmul_res738 to i32*
  %dim740 = getelementptr i32* %dim739, i32 -1
  %dim741 = load i32* %dim740
  %new742 = alloca i8, i32 %dim741
  call void @memset(i8* %new742, i32 0, i32 %dim741)
  %new743 = bitcast double* %mmul_res738 to i8*
  %new744 = getelementptr i8* %new743, i8 -16
  %42 = call i32 @memcpy(i8* %new742, i8* %new744, i32 %dim741)
  tail call void @free(i8* %new744)
  %new745 = getelementptr i8* %new742, i8 16
  %cp746 = bitcast i8* %new745 to double*
  %msub_res = call double* @msub(double* %y726, double* %cp746)
  %dim747 = bitcast double* %msub_res to i32*
  %dim748 = getelementptr i32* %dim747, i32 -1
  %dim749 = load i32* %dim748
  %new750 = alloca i8, i32 %dim749
```

```
call void @memset(i8* %new750, i32 0, i32 %dim749)
%new751 = bitcast double* %msub_res to i8*
%new752 = getelementptr i8* %new751, i8 -16
%43 = call i32 @memcpy(i8* %new750, i8* %new752, i32 %dim749)
tail call void @free(i8* %new752)
%new753 = getelementptr i8* %new750, i8 16
%cp754 = bitcast i8* %new753 to double*
%new755 = alloca i8, i32 16
call void @memset(i8* %new755, i32 0, i32 16)
%new756 = getelementptr i8* %new755, i8 16
%dim757 = bitcast i8* %new756 to i32*
%dim758 = getelementptr i32* %dim757, i32 -1
store i32 16, i32* %dim758
%dim759 = bitcast i8* %new756 to i32*
%dim760 = getelementptr i32* %dim759, i32 -2
store i32 0, i32* %dim760
%dim761 = bitcast i8* %new756 to i32*
%dim762 = getelementptr i32* %dim761, i32 -4
store i32 0, i32* %dim762
%dim763 = bitcast i8* %new756 to i32*
%dim764 = getelementptr i32* %dim763, i32 -3
store i32 0, i32* %dim764
%new765 = bitcast i8* %new756 to i32*
%dim766 = getelementptr i32* %new765, i32 -4
store i32 0, i32* %dim766
%dim767 = getelementptr i32* %new765, i32 -3
store i32 0, i32* %dim767
%new768 = bitcast i32* %new765 to double*
%new769 = alloca i8, i32 24
call void @memset(i8* %new769, i32 0, i32 24)
%new770 = getelementptr i8* %new769, i8 16
%dim771 = bitcast i8* %new770 to i32*
%dim772 = getelementptr i32* %dim771, i32 -1
store i32 24, i32* %dim772
%dim773 = bitcast i8* %new770 to i32*
%dim774 = getelementptr i32* %dim773, i32 -2
store i32 1, i32* %dim774
%dim775 = bitcast i8* %new770 to i32*
%dim776 = getelementptr i32* %dim775, i32 -4
store i32 0, i32* %dim776
%dim777 = bitcast i8* %new770 to i32*
%dim778 = getelementptr i32* %dim777, i32 -3
store i32 0, i32* %dim778
%new779 = bitcast i8* %new770 to i32*
%dim780 = getelementptr i32* %new779, i32 -4
store i32 1, i32* %dim780
%dim781 = getelementptr i32* %new779, i32 -3
store i32 1, i32* %dim781
%new782 = bitcast i32* %new779 to double*
%put783 = getelementptr double* %new782, i32 0
store double -1.000000e+00, double* %put783
%horzcat_res784 = call double* @horzcat(double* %new768, double* %new782)
%dim785 = bitcast double* %horzcat_res784 to i32*
%dim786 = getelementptr i32* %dim785, i32 -1
%dim787 = load i32* %dim786
%new788 = alloca i8, i32 %dim787
call void @memset(i8* %new788, i32 0, i32 %dim787)
%new789 = bitcast double* %horzcat_res784 to i8*
%new790 = getelementptr i8* %new789, i8 -16
%44 = call i32 @memcpy(i8* %new788, i8* %new790, i32 %dim787)
tail call void @free(i8* %new790)
```

```
%new791 = getelementptr i8* %new788, i8 16
%cp792 = bitcast i8* %new791 to double*
%vertcat_res793 = call double* @vertcat(double* %new768, double* %cp792)
%dim794 = bitcast double* %vertcat_res793 to i32*
%dim795 = getelementptr i32* %dim794, i32 -1
%dim796 = load i32* %dim795
%new797 = alloca i8, i32 %dim796
call void @memset(i8* %new797, i32 0, i32 %dim796)
%new798 = bitcast double* %vertcat_res793 to i8*
%new799 = getelementptr i8* %new798, i8 -16
%45 = call i32 @memcpy(i8* %new797, i8* %new799, i32 %dim796)
tail call void @free(i8* %new799)
%new800 = getelementptr i8* %new797, i8 16
%cp801 = bitcast i8* %new800 to double*
%mlt_res = call i32* @mlt(double* %cp754, double* %cp801)
%dim802 = getelementptr i32* %mlt_res, i32 -1
%dim803 = load i32* %dim802
%new804 = alloca i8, i32 %dim803
call void @memset(i8* %new804, i32 0, i32 %dim803)
%new805 = bitcast i32* %mlt_res to i8*
%new806 = getelementptr i8* %new805, i8 -16
%46 = call i32 @memcpy(i8* %new804, i8* %new806, i32 %dim803)
tail call void @free(i8* %new806)
%new807 = getelementptr i8* %new804, i8 16
%cp808 = bitcast i8* %new807 to i32*
store i32* %cp808, i32** %outliers
%new809 = alloca i8, i32 16
call void @memset(i8* %new809, i32 0, i32 16)
%new810 = getelementptr i8* %new809, i8 16
%dim811 = bitcast i8* %new810 to i32*
%dim812 = getelementptr i32* %dim811, i32 -1
store i32 16, i32* %dim812
%dim813 = bitcast i8* %new810 to i32*
%dim814 = getelementptr i32* %dim813, i32 -2
store i32 0, i32* %dim814
%dim815 = bitcast i8* %new810 to i32*
%dim816 = getelementptr i32* %dim815, i32 -4
store i32 0, i32* %dim816
%dim817 = bitcast i8* %new810 to i32*
%dim818 = getelementptr i32* %dim817, i32 -3
store i32 0, i32* %dim818
%new819 = bitcast i8* %new810 to i32*
%dim820 = getelementptr i32* %new819, i32 -4
store i32 0, i32* %dim820
%dim821 = getelementptr i32* %new819, i32 -3
store i32 0, i32* %dim821
%new822 = bitcast i32* %new819 to double*
%x823 = load double** %x
%y824 = load double** %y
%horzcat_res825 = call double* @horzcat(double* %new822, double* %y824)
%dim826 = bitcast double* %horzcat_res825 to i32*
%dim827 = getelementptr i32* %dim826, i32 -1
%dim828 = load i32* %dim827
%new829 = alloca i8, i32 %dim828
call void @memset(i8* %new829, i32 0, i32 %dim828)
%new830 = bitcast double* %horzcat_res825 to i8*
%new831 = getelementptr i8* %new830, i8 -16
%47 = call i32 @memcpy(i8* %new829, i8* %new831, i32 %dim828)
tail call void @free(i8* %new831)
%new832 = getelementptr i8* %new829, i8 16
%cp833 = bitcast i8* %new832 to double*
```

```
  %horzcat_res834 = call double* @horzcat(double* %cp833, double* %x823)
  %dim835 = bitcast double* %horzcat_res834 to i32*
  %dim836 = getelementptr i32* %dim835, i32 -1
  %dim837 = load i32* %dim836
  %new838 = alloca i8, i32 %dim837
  call void @memset(i8* %new838, i32 0, i32 %dim837)
  %new839 = bitcast double* %horzcat_res834 to i8*
  %new840 = getelementptr i8* %new839, i8 -16
  %48 = call i32 @memcpy(i8* %new838, i8* %new840, i32 %dim837)
  tail call void @free(i8* %new840)
  %new841 = getelementptr i8* %new838, i8 16
  %cp842 = bitcast i8* %new841 to double*
  %vertcat_res843 = call double* @vertcat(double* %new822, double* %cp842)
  %dim844 = bitcast double* %vertcat_res843 to i32*
  %dim845 = getelementptr i32* %dim844, i32 -1
  %dim846 = load i32* %dim845
  %new847 = alloca i8, i32 %dim846
  call void @memset(i8* %new847, i32 0, i32 %dim846)
  %new848 = bitcast double* %vertcat_res843 to i8*
  %new849 = getelementptr i8* %new848, i8 -16
  %49 = call i32 @memcpy(i8* %new847, i8* %new849, i32 %dim846)
  tail call void @free(i8* %new849)
  %new850 = getelementptr i8* %new847, i8 16
  %cp851 = bitcast i8* %new850 to double*
  call void @printmat(double* %cp851)
  call void @printstring(i8* getelementptr inbounds ([21 x i8]* @str, i32 0, i32 0))
  %outliers852 = load i32** %outliers
  %tmp = icmp eq i32* %outliers852, null
  %outliers853 = load i32** %outliers
  %dim854 = getelementptr i32* %outliers853, i32 -2
  %dim855 = load i32* %dim854
  %tmp856 = select i1 %tmp, i32 0, i32 %dim855
  call void @printint(i32 %tmp856)
  call void @println()
  ret i32 0
}

define double* @inv(double* %a) {
entry:
  %a1 = alloca double*
  store double* %a, double** %a1
  %a2 = load double** %a1
  %cofactor_res = call double* @cofactor(double* %a2)
  %dim = bitcast double* %cofactor_res to i32*
  %dim3 = getelementptr i32* %dim, i32 -1
  %dim4 = load i32* %dim3
  %new = alloca i8, i32 %dim4
  call void @memset(i8* %new, i32 0, i32 %dim4)
  %new5 = bitcast double* %cofactor_res to i8*
  %new6 = getelementptr i8* %new5, i8 -16
  %0 = call i32 @memcpy(i8* %new, i8* %new6, i32 %dim4)
  tail call void @free(i8* %new6)
  %new7 = getelementptr i8* %new, i8 16
  %cp = bitcast i8* %new7 to double*
  %mtransp_res = call double* @mtransp(double* %cp)
  %dim8 = bitcast double* %mtransp_res to i32*
  %dim9 = getelementptr i32* %dim8, i32 -1
  %dim10 = load i32* %dim9
  %new11 = alloca i8, i32 %dim10
  call void @memset(i8* %new11, i32 0, i32 %dim10)
  %new12 = bitcast double* %mtransp_res to i8*
  %new13 = getelementptr i8* %new12, i8 -16
  %1 = call i32 @memcpy(i8* %new11, i8* %new13, i32 %dim10)
  tail call void @free(i8* %new13)
  %new14 = getelementptr i8* %new11, i8 16
  %cp15 = bitcast i8* %new14 to double*
  %new16 = alloca i8, i32 16
  call void @memset(i8* %new16, i32 0, i32 16)
  %new17 = getelementptr i8* %new16, i8 16
  %dim18 = bitcast i8* %new17 to i32*
  %dim19 = getelementptr i32* %dim18, i32 -1
  store i32 16, i32* %dim19
  %dim20 = bitcast i8* %new17 to i32*
  %dim21 = getelementptr i32* %dim20, i32 -2
  store i32 0, i32* %dim21
  %dim22 = bitcast i8* %new17 to i32*
  %dim23 = getelementptr i32* %dim22, i32 -4
  store i32 0, i32* %dim23
  %dim24 = bitcast i8* %new17 to i32*
  %dim25 = getelementptr i32* %dim24, i32 -3
  store i32 0, i32* %dim25
  %new26 = bitcast i8* %new17 to i32*
  %dim27 = getelementptr i32* %new26, i32 -4
  store i32 0, i32* %dim27
  %dim28 = getelementptr i32* %new26, i32 -3
  store i32 0, i32* %dim28
  %new29 = bitcast i32* %new26 to double*
  %a30 = load double** %a1
  %det_res = call double @det(double* %a30)
  %new31 = alloca i8, i32 24
  call void @memset(i8* %new31, i32 0, i32 24)
  %new32 = getelementptr i8* %new31, i8 16
  %dim33 = bitcast i8* %new32 to i32*
  %dim34 = getelementptr i32* %dim33, i32 -1
  store i32 24, i32* %dim34
  %dim35 = bitcast i8* %new32 to i32*
  %dim36 = getelementptr i32* %dim35, i32 -2
  store i32 1, i32* %dim36
  %dim37 = bitcast i8* %new32 to i32*
  %dim38 = getelementptr i32* %dim37, i32 -4
  store i32 0, i32* %dim38
  %dim39 = bitcast i8* %new32 to i32*
  %dim40 = getelementptr i32* %dim39, i32 -3
  store i32 0, i32* %dim40
  %new41 = bitcast i8* %new32 to i32*
  %dim42 = getelementptr i32* %new41, i32 -4
  store i32 1, i32* %dim42
  %dim43 = getelementptr i32* %new41, i32 -3
  store i32 1, i32* %dim43
  %new44 = bitcast i32* %new41 to double*
  %put = getelementptr double* %new44, i32 0
  store double %det_res, double* %put
  %horzcat_res = call double* @horzcat(double* %new29, double* %new44)
  %dim45 = bitcast double* %horzcat_res to i32*
  %dim46 = getelementptr i32* %dim45, i32 -1
  %dim47 = load i32* %dim46
  %new48 = alloca i8, i32 %dim47
  call void @memset(i8* %new48, i32 0, i32 %dim47)
  %new49 = bitcast double* %horzcat_res to i8*
  %new50 = getelementptr i8* %new49, i8 -16
  %2 = call i32 @memcpy(i8* %new48, i8* %new50, i32 %dim47)
  tail call void @free(i8* %new50)
  %new51 = getelementptr i8* %new48, i8 16
  %cp52 = bitcast i8* %new51 to double*
  %vertcat_res = call double* @vertcat(double* %new29, double* %cp52)
  %dim53 = bitcast double* %vertcat_res to i32*
  %dim54 = getelementptr i32* %dim53, i32 -1
  %dim55 = load i32* %dim54
  %new56 = alloca i8, i32 %dim55
  call void @memset(i8* %new56, i32 0, i32 %dim55)
  %new57 = bitcast double* %vertcat_res to i8*
  %new58 = getelementptr i8* %new57, i8 -16
  %3 = call i32 @memcpy(i8* %new56, i8* %new58, i32 %dim55)
  tail call void @free(i8* %new58)
  %new59 = getelementptr i8* %new56, i8 16
  %cp60 = bitcast i8* %new59 to double*
  %mdotdiv_res = call double* @mdotdiv(double* %cp15, double* %cp60)
  %dim61 = bitcast double* %mdotdiv_res to i32*
  %dim62 = getelementptr i32* %dim61, i32 -1
  %dim63 = load i32* %dim62
  %new64 = alloca i8, i32 %dim63
  call void @memset(i8* %new64, i32 0, i32 %dim63)
  %new65 = bitcast double* %mdotdiv_res to i8*
  %new66 = getelementptr i8* %new65, i8 -16
  %4 = call i32 @memcpy(i8* %new64, i8* %new66, i32 %dim63)
  tail call void @free(i8* %new66)
  %new67 = getelementptr i8* %new64, i8 16
  %cp68 = bitcast i8* %new67 to double*
  %dim69 = bitcast double* %cp68 to i32*
  %dim70 = getelementptr i32* %dim69, i32 -1
  %dim71 = load i32* %dim70
  %mallocsize = mul i32 %dim71, ptrtoint (i8* getelementptr (i8* null, i32 1) to i32)
  %new72 = tail call i8* @malloc(i32 %mallocsize)
  call void @memset(i8* %new72, i32 0, i32 %dim71)
  %new73 = bitcast double* %cp68 to i8*
  %new74 = getelementptr i8* %new73, i8 -16
  %5 = call i32 @memcpy(i8* %new72, i8* %new74, i32 %dim71)
  %new75 = getelementptr i8* %new72, i8 16
  %cp76 = bitcast i8* %new75 to double*
  ret double* %cp76
}

define double* @cofactor(double* %a) {
entry:
  %a1 = alloca double*
  store double* %a, double** %a1
  %i = alloca i32
  store i32 0, i32* %i
  %j = alloca i32
  store i32 0, i32* %j
  %ii = alloca i32
  store i32 0, i32* %ii
  %jj = alloca i32
  store i32 0, i32* %jj
  %i1 = alloca i32
  store i32 0, i32* %i1
  %j1 = alloca i32
  store i32 0, i32* %j1
  %det = alloca double
  store double 0.000000e+00, double* %det
  %c = alloca double*
  store double* null, double** %c
  %n = alloca i32
```

```
store i32 0, i32* %n
%b = alloca double*
store double* null, double** %b
%a2 = load double** %a1
call void @checkmatsquare(double* %a2)
%a3 = load double** %a1
%rows_res = call i32 @rows(double* %a3)
store i32 %rows_res, i32* %n
%n4 = load i32* %n
%n5 = load i32* %n
%new = mul i32 %n5, %n4
%new6 = mul i32 %new, 8
%new7 = add i32 %new6, 16
%new8 = alloca i8, i32 %new7
call void @memset(i8* %new8, i32 0, i32 %new7)
%new9 = getelementptr i8* %new8, i8 16
%dim = bitcast i8* %new9 to i32*
%dim10 = getelementptr i32* %dim, i32 -1
store i32 %new7, i32* %dim10
%dim11 = bitcast i8* %new9 to i32*
%dim12 = getelementptr i32* %dim11, i32 -2
store i32 %new, i32* %dim12
%dim13 = bitcast i8* %new9 to i32*
%dim14 = getelementptr i32* %dim13, i32 -4
store i32 0, i32* %dim14
%dim15 = bitcast i8* %new9 to i32*
%dim16 = getelementptr i32* %dim15, i32 -3
store i32 0, i32* %dim16
%new17 = bitcast i8* %new9 to i32*
%dim18 = getelementptr i32* %new17, i32 -4
store i32 %n5, i32* %dim18
%dim19 = getelementptr i32* %new17, i32 -3
store i32 %n4, i32* %dim19
%new20 = bitcast i32* %new17 to double*
store double* %new20, double** %b
%n21 = load i32* %n
%tmp = sub i32 %n21, 1
%n22 = load i32* %n
%tmp23 = sub i32 %n22, 1
%new24 = mul i32 %tmp23, %tmp
%new25 = mul i32 %new24, 8
%new26 = add i32 %new25, 16
%new27 = alloca i8, i32 %new26
call void @memset(i8* %new27, i32 0, i32 %new26)
%new28 = getelementptr i8* %new27, i8 16
%dim29 = bitcast i8* %new28 to i32*
%dim30 = getelementptr i32* %dim29, i32 -1
store i32 %new26, i32* %dim30
%dim31 = bitcast i8* %new28 to i32*
%dim32 = getelementptr i32* %dim31, i32 -2
store i32 %new24, i32* %dim32
%dim33 = bitcast i8* %new28 to i32*
%dim34 = getelementptr i32* %dim33, i32 -4
store i32 0, i32* %dim34
%dim35 = bitcast i8* %new28 to i32*
%dim36 = getelementptr i32* %dim35, i32 -3
store i32 0, i32* %dim36
%new37 = bitcast i8* %new28 to i32*
%dim38 = getelementptr i32* %new37, i32 -4
store i32 %tmp23, i32* %dim38
%dim39 = getelementptr i32* %new37, i32 -3
```

```
  store i32 %tmp, i32* %dim39
  %new40 = bitcast i32* %new37 to double*
  store double* %new40, double** %c
  store i32 0, i32* %j
  br label %while

while: ; preds = %merge171, %entry
  %j174 = load i32* %j
  %n175 = load i32* %n
  %tmp176 = icmp slt i32 %j174, %n175
  br i1 %tmp176, label %while_body, label %merge177

while_body: ; preds = %while
  store i32 0, i32* %i
  br label %while41

while41: ; preds = %merge104, %while_body
  %i168 = load i32* %i
  %n169 = load i32* %n
  %tmp170 = icmp slt i32 %i168, %n169
  br i1 %tmp170, label %while_body42, label %merge171

while_body42: ; preds = %while41
  store i32 0, i32* %i1
  store i32 0, i32* %ii
  br label %while43

while43: ; preds = %merge, %while_body42
  %ii101 = load i32* %ii
  %n102 = load i32* %n
  %tmp103 = icmp slt i32 %ii101, %n102
  br i1 %tmp103, label %while_body44, label %merge104

while_body44: ; preds = %while43
  %ii45 = load i32* %ii
  %i46 = load i32* %i
  %tmp47 = icmp ne i32 %ii45, %i46
  br i1 %tmp47, label %then, label %else98

merge: ; preds = %else98, %merge95
  %ii99 = load i32* %ii
  %tmp100 = add i32 %ii99, 1
  store i32 %tmp100, i32* %ii
  br label %while43

then: ; preds = %while_body44
  store i32 0, i32* %j1
  store i32 0, i32* %jj
  br label %while48

while48: ; preds = %merge53, %then
  %jj92 = load i32* %jj
  %n93 = load i32* %n
  %tmp94 = icmp slt i32 %jj92, %n93
  br i1 %tmp94, label %while_body49, label %merge95

while_body49: ; preds = %while48
  %jj50 = load i32* %jj
  %j51 = load i32* %j
  %tmp52 = icmp ne i32 %jj50, %j51
  br i1 %tmp52, label %then54, label %else
```

```
merge53: ; preds = %else, %then54
  %jj90 = load i32* %jj
  %tmp91 = add i32 %jj90, 1
  store i32 %tmp91, i32* %jj
  br label %while48

then54: ; preds = %while_body49
  %i155 = load i32* %i1
  %j156 = load i32* %j1
  %c57 = load double** %c
  %ii58 = load i32* %ii
  %jj59 = load i32* %jj
  %a60 = load double** %a1
  call void @checkmatrc(double* %a60, i32 %ii58, i32 %jj59)
  %dim61 = bitcast double* %a60 to i32*
  %dim62 = getelementptr i32* %dim61, i32 -3
  %dim63 = load i32* %dim62
  %get = mul i32 %ii58, %dim63
  %get64 = add i32 %jj59, %get
  %get65 = getelementptr double* %a60, i32 %get64
  %get66 = load double* %get65
  %new67 = alloca i8, i32 24
  call void @memset(i8* %new67, i32 0, i32 24)
  %new68 = getelementptr i8* %new67, i8 16
  %dim69 = bitcast i8* %new68 to i32*
  %dim70 = getelementptr i32* %dim69, i32 -1
  store i32 24, i32* %dim70
  %dim71 = bitcast i8* %new68 to i32*
  %dim72 = getelementptr i32* %dim71, i32 -2
  store i32 1, i32* %dim72
  %dim73 = bitcast i8* %new68 to i32*
  %dim74 = getelementptr i32* %dim73, i32 -4
  store i32 0, i32* %dim74
  %dim75 = bitcast i8* %new68 to i32*
  %dim76 = getelementptr i32* %dim75, i32 -3
  store i32 0, i32* %dim76
  %new77 = bitcast i8* %new68 to i32*
  %dim78 = getelementptr i32* %new77, i32 -4
  store i32 1, i32* %dim78
  %dim79 = getelementptr i32* %new77, i32 -3
  store i32 1, i32* %dim79
  %new80 = bitcast i32* %new77 to double*
  %put = getelementptr double* %new80, i32 0
  store double %get66, double* %put
  call void @checkmatrc(double* %c57, i32 %i155, i32 %j156)
  call void @checkmatscalar(double* %new80)
  %get81 = getelementptr double* %new80, i32 0
  %get82 = load double* %get81
  %dim83 = bitcast double* %c57 to i32*
  %dim84 = getelementptr i32* %dim83, i32 -3
  %dim85 = load i32* %dim84
  %getrc = mul i32 %i155, %dim85
  %getrc86 = add i32 %j156, %getrc
  %put87 = getelementptr double* %c57, i32 %getrc86
  store double %get82, double* %put87
  %j188 = load i32* %j1
  %tmp89 = add i32 %j188, 1
  store i32 %tmp89, i32* %j1
  br label %merge53
```

```
else: ; preds = %while_body49
  br label %merge53

merge95: ; preds = %while48
  %i196 = load i32* %i1
  %tmp97 = add i32 %i196, 1
  store i32 %tmp97, i32* %i1
  br label %merge

else98: ; preds = %while_body44
  br label %merge

merge104: ; preds = %while43
  %i105 = load i32* %i
  %j106 = load i32* %j
  %b107 = load double** %b
  %new108 = alloca i8, i32 16
  call void @memset(i8* %new108, i32 0, i32 16)
  %new109 = getelementptr i8* %new108, i8 16
  %dim110 = bitcast i8* %new109 to i32*
  %dim111 = getelementptr i32* %dim110, i32 -1
  store i32 16, i32* %dim111
  %dim112 = bitcast i8* %new109 to i32*
  %dim113 = getelementptr i32* %dim112, i32 -2
  store i32 0, i32* %dim113
  %dim114 = bitcast i8* %new109 to i32*
  %dim115 = getelementptr i32* %dim114, i32 -4
  store i32 0, i32* %dim115
  %dim116 = bitcast i8* %new109 to i32*
  %dim117 = getelementptr i32* %dim116, i32 -3
  store i32 0, i32* %dim117
  %new118 = bitcast i8* %new109 to i32*
  %dim119 = getelementptr i32* %new118, i32 -4
  store i32 0, i32* %dim119
  %dim120 = getelementptr i32* %new118, i32 -3
  store i32 0, i32* %dim120
  %new121 = bitcast i32* %new118 to double*
  %i122 = load i32* %i
  %j123 = load i32* %j
  %tmp124 = add i32 %i122, %j123
  %float_of = sitofp i32 %tmp124 to double
  %tmp125 = fadd double %float_of, 2.000000e+00
  %pow_res = call double @pow(double -1.000000e+00, double %tmp125)
  %c126 = load double** %c
  %det_res = call double @det(double* %c126)
  %tmp127 = fmul double %pow_res, %det_res
  %new128 = alloca i8, i32 24
  call void @memset(i8* %new128, i32 0, i32 24)
  %new129 = getelementptr i8* %new128, i8 16
  %dim130 = bitcast i8* %new129 to i32*
  %dim131 = getelementptr i32* %dim130, i32 -1
  store i32 24, i32* %dim131
  %dim132 = bitcast i8* %new129 to i32*
  %dim133 = getelementptr i32* %dim132, i32 -2
  store i32 1, i32* %dim133
  %dim134 = bitcast i8* %new129 to i32*
  %dim135 = getelementptr i32* %dim134, i32 -4
  store i32 0, i32* %dim135
  %dim136 = bitcast i8* %new129 to i32*
  %dim137 = getelementptr i32* %dim136, i32 -3
  store i32 0, i32* %dim137
  %new138 = bitcast i8* %new129 to i32*
  %dim139 = getelementptr i32* %new138, i32 -4
  store i32 1, i32* %dim139
  %dim140 = getelementptr i32* %new138, i32 -3
  store i32 1, i32* %dim140
  %new141 = bitcast i32* %new138 to double*
  %put142 = getelementptr double* %new141, i32 0
  store double %tmp127, double* %put142
  %horzcat_res = call double* @horzcat(double* %new121, double* %new141)
  %dim143 = bitcast double* %horzcat_res to i32*
  %dim144 = getelementptr i32* %dim143, i32 -1
  %dim145 = load i32* %dim144
  %new146 = alloca i8, i32 %dim145
  call void @memset(i8* %new146, i32 0, i32 %dim145)
  %new147 = bitcast double* %horzcat_res to i8*
  %new148 = getelementptr i8* %new147, i8 -16
  %0 = call i32 @memcpy(i8* %new146, i8* %new148, i32 %dim145)
  tail call void @free(i8* %new148)
  %new149 = getelementptr i8* %new146, i8 16
  %cp = bitcast i8* %new149 to double*
  %vertcat_res = call double* @vertcat(double* %new121, double* %cp)
  %dim150 = bitcast double* %vertcat_res to i32*
  %dim151 = getelementptr i32* %dim150, i32 -1
  %dim152 = load i32* %dim151
  %new153 = alloca i8, i32 %dim152
  call void @memset(i8* %new153, i32 0, i32 %dim152)
  %new154 = bitcast double* %vertcat_res to i8*
  %new155 = getelementptr i8* %new154, i8 -16
  %1 = call i32 @memcpy(i8* %new153, i8* %new155, i32 %dim152)
  tail call void @free(i8* %new155)
  %new156 = getelementptr i8* %new153, i8 16
  %cp157 = bitcast i8* %new156 to double*
  call void @checkmatrc(double* %b107, i32 %i105, i32 %j106)
  call void @checkmatscalar(double* %cp157)
  %get158 = getelementptr double* %cp157, i32 0
  %get159 = load double* %get158
  %dim160 = bitcast double* %b107 to i32*
  %dim161 = getelementptr i32* %dim160, i32 -3
  %dim162 = load i32* %dim161
  %getrc163 = mul i32 %i105, %dim162
  %getrc164 = add i32 %j106, %getrc163
  %put165 = getelementptr double* %b107, i32 %getrc164
  store double %get159, double* %put165
  %i166 = load i32* %i
  %tmp167 = add i32 %i166, 1
  store i32 %tmp167, i32* %i
  br label %while41

merge171: ; preds = %while41
  %j172 = load i32* %j
  %tmp173 = add i32 %j172, 1
  store i32 %tmp173, i32* %j
  br label %while

merge177: ; preds = %while
  %b178 = load double** %b
  %dim179 = bitcast double* %b178 to i32*
  %dim180 = getelementptr i32* %dim179, i32 -1
  %dim181 = load i32* %dim180
  %mallocsize = mul i32 %dim181, ptrtoint (i8* getelementptr (i8* null, i32 1) to i32)
  %new182 = tail call i8* @malloc(i32 %mallocsize)
  call void @memset(i8* %new182, i32 0, i32 %dim181)
  %new183 = bitcast double* %b178 to i8*
  %new184 = getelementptr i8* %new183, i8 -16
  %2 = call i32 @memcpy(i8* %new182, i8* %new184, i32 %dim181)
  %new185 = getelementptr i8* %new182, i8 16
  %cp186 = bitcast i8* %new185 to double*
  ret double* %cp186
}

define double @det(double* %a) {
entry:
  %a1 = alloca double*
  store double* %a, double** %a1
  %det = alloca double*
  store double* null, double** %det
  %i = alloca i32
  store i32 0, i32* %i
  %j = alloca i32
  store i32 0, i32* %j
  %j1 = alloca i32
  store i32 0, i32* %j1
  %j2 = alloca i32
  store i32 0, i32* %j2
  %m = alloca double*
  store double* null, double** %m
  %tmp = alloca double
  store double 0.000000e+00, double* %tmp
  %a2 = load double** %a1
  call void @checkmatsquare(double* %a2)
  %a3 = load double** %a1
  %rows_res = call i32 @rows(double* %a3)
  %tmp4 = icmp eq i32 %rows_res, 1
  br i1 %tmp4, label %then, label %else

merge: ; preds = %merge27, %then
  %det446 = load double** %det
  call void @checkmatscalar(double* %det446)
  %get447 = getelementptr double* %det446, i32 0
  %get448 = load double* %get447
  ret double %get448

then: ; preds = %entry
  %a5 = load double** %a1
  call void @checkmatrc(double* %a5, i32 0, i32 0)
  %dim = bitcast double* %a5 to i32*
  %dim6 = getelementptr i32* %dim, i32 -3
  %dim7 = load i32* %dim6
  %get = mul i32 0, %dim7
  %get8 = add i32 0, %get
  %get9 = getelementptr double* %a5, i32 %get8
  %get10 = load double* %get9
  %new = alloca i8, i32 24
  call void @memset(i8* %new, i32 0, i32 24)
  %new11 = getelementptr i8* %new, i8 16
  %dim12 = bitcast i8* %new11 to i32*
  %dim13 = getelementptr i32* %dim12, i32 -1
  store i32 24, i32* %dim13
  %dim14 = bitcast i8* %new11 to i32*
  %dim15 = getelementptr i32* %dim14, i32 -2
  store i32 1, i32* %dim15
  %dim16 = bitcast i8* %new11 to i32*
  %dim17 = getelementptr i32* %dim16, i32 -4
  store i32 0, i32* %dim17
  %dim18 = bitcast i8* %new11 to i32*
  %dim19 = getelementptr i32* %dim18, i32 -3
  store i32 0, i32* %dim19
  %new20 = bitcast i8* %new11 to i32*
  %dim21 = getelementptr i32* %new20, i32 -4
  store i32 1, i32* %dim21
  %dim22 = getelementptr i32* %new20, i32 -3
  store i32 1, i32* %dim22
  %new23 = bitcast i32* %new20 to double*
  %put = getelementptr double* %new23, i32 0
  store double %get10, double* %put
  store double* %new23, double** %det
  br label %merge

else: ; preds = %entry
  %a24 = load double** %a1
  %rows_res25 = call i32 @rows(double* %a24)
  %tmp26 = icmp eq i32 %rows_res25, 2
  br i1 %tmp26, label %then28, label %else145

merge27: ; preds = %merge445, %then28
  br label %merge

then28: ; preds = %else
  %a29 = load double** %a1
  call void @checkmatrc(double* %a29, i32 0, i32 0)
  %dim30 = bitcast double* %a29 to i32*
  %dim31 = getelementptr i32* %dim30, i32 -3
  %dim32 = load i32* %dim31
  %get33 = mul i32 0, %dim32
  %get34 = add i32 0, %get33
  %get35 = getelementptr double* %a29, i32 %get34
  %get36 = load double* %get35
  %new37 = alloca i8, i32 24
  call void @memset(i8* %new37, i32 0, i32 24)
  %new38 = getelementptr i8* %new37, i8 16
  %dim39 = bitcast i8* %new38 to i32*
  %dim40 = getelementptr i32* %dim39, i32 -1
  store i32 24, i32* %dim40
  %dim41 = bitcast i8* %new38 to i32*
  %dim42 = getelementptr i32* %dim41, i32 -2
  store i32 1, i32* %dim42
  %dim43 = bitcast i8* %new38 to i32*
  %dim44 = getelementptr i32* %dim43, i32 -4
  store i32 0, i32* %dim44
  %dim45 = bitcast i8* %new38 to i32*
  %dim46 = getelementptr i32* %dim45, i32 -3
  store i32 0, i32* %dim46
  %new47 = bitcast i8* %new38 to i32*
  %dim48 = getelementptr i32* %new47, i32 -4
  store i32 1, i32* %dim48
  %dim49 = getelementptr i32* %new47, i32 -3
  store i32 1, i32* %dim49
  %new50 = bitcast i32* %new47 to double*
  %put51 = getelementptr double* %new50, i32 0
  store double %get36, double* %put51
  %a52 = load double** %a1
  call void @checkmatrc(double* %a52, i32 1, i32 1)
  %dim53 = bitcast double* %a52 to i32*
```

```
%dim54 = getelementptr i32* %dim53, i32 -3
%dim55 = load i32* %dim54
%get56 = mul i32 1, %dim55
%get57 = add i32 1, %get56
%get58 = getelementptr double* %a52, i32 %get57
%get59 = load double* %get58
%new60 = alloca i8, i32 24
call void @memset(i8* %new60, i32 0, i32 24)
%new61 = getelementptr i8* %new60, i8 16
%dim62 = bitcast i8* %new61 to i32*
%dim63 = getelementptr i32* %dim62, i32 -1
store i32 24, i32* %dim63
%dim64 = bitcast i8* %new61 to i32*
%dim65 = getelementptr i32* %dim64, i32 -2
store i32 1, i32* %dim65
%dim66 = bitcast i8* %new61 to i32*
%dim67 = getelementptr i32* %dim66, i32 -4
store i32 0, i32* %dim67
%dim68 = bitcast i8* %new61 to i32*
%dim69 = getelementptr i32* %dim68, i32 -3
store i32 0, i32* %dim69
%new70 = bitcast i8* %new61 to i32*
%dim71 = getelementptr i32* %new70, i32 -4
store i32 1, i32* %dim71
%dim72 = getelementptr i32* %new70, i32 -3
store i32 1, i32* %dim72
%new73 = bitcast i32* %new70 to double*
%put74 = getelementptr double* %new73, i32 0
store double %get59, double* %put74
%mmul_res = call double* @mmul(double* %new50, double* %new73)
%dim75 = bitcast double* %mmul_res to i32*
%dim76 = getelementptr i32* %dim75, i32 -1
%dim77 = load i32* %dim76
%new78 = alloca i8, i32 %dim77
call void @memset(i8* %new78, i32 0, i32 %dim77)
%new79 = bitcast double* %mmul_res to i8*
%new80 = getelementptr i8* %new79, i8 -16
%0 = call i32 @memcpy(i8* %new78, i8* %new80, i32 %dim77)
tail call void @free(i8* %new80)
%new81 = getelementptr i8* %new78, i8 16
%cp = bitcast i8* %new81 to double*
%a82 = load double** %a1
call void @checkmatrc(double* %a82, i32 0, i32 1)
%dim83 = bitcast double* %a82 to i32*
%dim84 = getelementptr i32* %dim83, i32 -3
%dim85 = load i32* %dim84
%get86 = mul i32 0, %dim85
%get87 = add i32 1, %get86
%get88 = getelementptr double* %a82, i32 %get87
%get89 = load double* %get88
%new90 = alloca i8, i32 24
call void @memset(i8* %new90, i32 0, i32 24)
%new91 = getelementptr i8* %new90, i8 16
%dim92 = bitcast i8* %new91 to i32*
%dim93 = getelementptr i32* %dim92, i32 -1
store i32 24, i32* %dim93
%dim94 = bitcast i8* %new91 to i32*
%dim95 = getelementptr i32* %dim94, i32 -2
store i32 1, i32* %dim95
%dim96 = bitcast i8* %new91 to i32*
%dim97 = getelementptr i32* %dim96, i32 -4
```

```
  store i32 0, i32* %dim97
  %dim98 = bitcast i8* %new91 to i32*
  %dim99 = getelementptr i32* %dim98, i32 -3
  store i32 0, i32* %dim99
  %new100 = bitcast i8* %new91 to i32*
  %dim101 = getelementptr i32* %new100, i32 -4
  store i32 1, i32* %dim101
  %dim102 = getelementptr i32* %new100, i32 -3
  store i32 1, i32* %dim102
  %new103 = bitcast i32* %new100 to double*
  %put104 = getelementptr double* %new103, i32 0
  store double %get89, double* %put104
  %a105 = load double** %a1
  call void @checkmatrc(double* %a105, i32 1, i32 0)
  %dim106 = bitcast double* %a105 to i32*
  %dim107 = getelementptr i32* %dim106, i32 -3
  %dim108 = load i32* %dim107
  %get109 = mul i32 1, %dim108
  %get110 = add i32 0, %get109
  %get111 = getelementptr double* %a105, i32 %get110
  %get112 = load double* %get111
  %new113 = alloca i8, i32 24
  call void @memset(i8* %new113, i32 0, i32 24)
  %new114 = getelementptr i8* %new113, i8 16
  %dim115 = bitcast i8* %new114 to i32*
  %dim116 = getelementptr i32* %dim115, i32 -1
  store i32 24, i32* %dim116
  %dim117 = bitcast i8* %new114 to i32*
  %dim118 = getelementptr i32* %dim117, i32 -2
  store i32 1, i32* %dim118
  %dim119 = bitcast i8* %new114 to i32*
  %dim120 = getelementptr i32* %dim119, i32 -4
  store i32 0, i32* %dim120
  %dim121 = bitcast i8* %new114 to i32*
  %dim122 = getelementptr i32* %dim121, i32 -3
  store i32 0, i32* %dim122
  %new123 = bitcast i8* %new114 to i32*
  %dim124 = getelementptr i32* %new123, i32 -4
  store i32 1, i32* %dim124
  %dim125 = getelementptr i32* %new123, i32 -3
  store i32 1, i32* %dim125
  %new126 = bitcast i32* %new123 to double*
  %put127 = getelementptr double* %new126, i32 0
  store double %get112, double* %put127
  %mmul_res128 = call double* @mmul(double* %new103, double* %new126)
  %dim129 = bitcast double* %mmul_res128 to i32*
  %dim130 = getelementptr i32* %dim129, i32 -1
  %dim131 = load i32* %dim130
  %new132 = alloca i8, i32 %dim131
  call void @memset(i8* %new132, i32 0, i32 %dim131)
  %new133 = bitcast double* %mmul_res128 to i8*
  %new134 = getelementptr i8* %new133, i8 -16
  %1 = call i32 @memcpy(i8* %new132, i8* %new134, i32 %dim131)
  tail call void @free(i8* %new134)
  %new135 = getelementptr i8* %new132, i8 16
  %cp136 = bitcast i8* %new135 to double*
  %msub_res = call double* @msub(double* %cp, double* %cp136)
  %dim137 = bitcast double* %msub_res to i32*
  %dim138 = getelementptr i32* %dim137, i32 -1
  %dim139 = load i32* %dim138
  %new140 = alloca i8, i32 %dim139
  call void @memset(i8* %new140, i32 0, i32 %dim139)
  %new141 = bitcast double* %msub_res to i8*
  %new142 = getelementptr i8* %new141, i8 -16
  %2 = call i32 @memcpy(i8* %new140, i8* %new142, i32 %dim139)
  tail call void @free(i8* %new142)
  %new143 = getelementptr i8* %new140, i8 16
  %cp144 = bitcast i8* %new143 to double*
  store double* %cp144, double** %det
  br label %merge27

else145: ; preds = %else
  %new146 = alloca i8, i32 16
  call void @memset(i8* %new146, i32 0, i32 16)
  %new147 = getelementptr i8* %new146, i8 16
  %dim148 = bitcast i8* %new147 to i32*
  %dim149 = getelementptr i32* %dim148, i32 -1
  store i32 16, i32* %dim149
  %dim150 = bitcast i8* %new147 to i32*
  %dim151 = getelementptr i32* %dim150, i32 -2
  store i32 0, i32* %dim151
  %dim152 = bitcast i8* %new147 to i32*
  %dim153 = getelementptr i32* %dim152, i32 -4
  store i32 0, i32* %dim153
  %dim154 = bitcast i8* %new147 to i32*
  %dim155 = getelementptr i32* %dim154, i32 -3
  store i32 0, i32* %dim155
  %new156 = bitcast i8* %new147 to i32*
  %dim157 = getelementptr i32* %new156, i32 -4
  store i32 0, i32* %dim157
  %dim158 = getelementptr i32* %new156, i32 -3
  store i32 0, i32* %dim158
  %new159 = bitcast i32* %new156 to double*
  %new160 = alloca i8, i32 24
  call void @memset(i8* %new160, i32 0, i32 24)
  %new161 = getelementptr i8* %new160, i8 16
  %dim162 = bitcast i8* %new161 to i32*
  %dim163 = getelementptr i32* %dim162, i32 -1
  store i32 24, i32* %dim163
  %dim164 = bitcast i8* %new161 to i32*
  %dim165 = getelementptr i32* %dim164, i32 -2
  store i32 1, i32* %dim165
  %dim166 = bitcast i8* %new161 to i32*
  %dim167 = getelementptr i32* %dim166, i32 -4
  store i32 0, i32* %dim167
  %dim168 = bitcast i8* %new161 to i32*
  %dim169 = getelementptr i32* %dim168, i32 -3
  store i32 0, i32* %dim169
  %new170 = bitcast i8* %new161 to i32*
  %dim171 = getelementptr i32* %new170, i32 -4
  store i32 1, i32* %dim171
  %dim172 = getelementptr i32* %new170, i32 -3
  store i32 1, i32* %dim172
  %new173 = bitcast i32* %new170 to double*
  %put174 = getelementptr double* %new173, i32 0
  store double 0.000000e+00, double* %put174
  %horzcat_res = call double* @horzcat(double* %new159, double* %new173)
  %dim175 = bitcast double* %horzcat_res to i32*
  %dim176 = getelementptr i32* %dim175, i32 -1
  %dim177 = load i32* %dim176
  %new178 = alloca i8, i32 %dim177
  call void @memset(i8* %new178, i32 0, i32 %dim177)
  %new179 = bitcast double* %horzcat_res to i8*
  %new180 = getelementptr i8* %new179, i8 -16
  %3 = call i32 @memcpy(i8* %new178, i8* %new180, i32 %dim177)
  tail call void @free(i8* %new180)
  %new181 = getelementptr i8* %new178, i8 16
  %cp182 = bitcast i8* %new181 to double*
  %vertcat_res = call double* @vertcat(double* %new159, double* %cp182)
  %dim183 = bitcast double* %vertcat_res to i32*
  %dim184 = getelementptr i32* %dim183, i32 -1
  %dim185 = load i32* %dim184
  %new186 = alloca i8, i32 %dim185
  call void @memset(i8* %new186, i32 0, i32 %dim185)
  %new187 = bitcast double* %vertcat_res to i8*
  %new188 = getelementptr i8* %new187, i8 -16
  %4 = call i32 @memcpy(i8* %new186, i8* %new188, i32 %dim185)
  tail call void @free(i8* %new188)
  %new189 = getelementptr i8* %new186, i8 16
  %cp190 = bitcast i8* %new189 to double*
  store double* %cp190, double** %det
  store i32 0, i32* %j1
  br label %while

while: ; preds = %merge285, %else145
  %j1436 = load i32* %j1
  %a437 = load double** %a1
  %tmp438 = icmp eq double* %a437, null
  %a439 = load double** %a1
  %dim440 = bitcast double* %a439 to i32*
  %dim441 = getelementptr i32* %dim440, i32 -3
  %dim442 = load i32* %dim441
  %tmp443 = select i1 %tmp438, i32 0, i32 %dim442
  %tmp444 = icmp slt i32 %j1436, %tmp443
  br i1 %tmp444, label %while_body, label %merge445

while_body: ; preds = %while
  %a191 = load double** %a1
  %tmp192 = icmp eq double* %a191, null
  %a193 = load double** %a1
  %dim194 = bitcast double* %a193 to i32*
  %dim195 = getelementptr i32* %dim194, i32 -3
  %dim196 = load i32* %dim195
  %tmp197 = select i1 %tmp192, i32 0, i32 %dim196
  %tmp198 = sub i32 %tmp197, 1
  %a199 = load double** %a1
  %rows_res200 = call i32 @rows(double* %a199)
  %tmp201 = sub i32 %rows_res200, 1
  %new202 = mul i32 %tmp201, %tmp198
  %new203 = mul i32 %new202, 8
  %new204 = add i32 %new203, 16
  %new205 = alloca i8, i32 %new204
  call void @memset(i8* %new205, i32 0, i32 %new204)
  %new206 = getelementptr i8* %new205, i8 16
  %dim207 = bitcast i8* %new206 to i32*
  %dim208 = getelementptr i32* %dim207, i32 -1
  store i32 %new204, i32* %dim208
  %dim209 = bitcast i8* %new206 to i32*
  %dim210 = getelementptr i32* %dim209, i32 -2
  store i32 %new202, i32* %dim210
  %dim211 = bitcast i8* %new206 to i32*
  %dim212 = getelementptr i32* %dim211, i32 -4
  store i32 0, i32* %dim212
  %dim213 = bitcast i8* %new206 to i32*
  %dim214 = getelementptr i32* %dim213, i32 -3
  store i32 0, i32* %dim214
  %new215 = bitcast i8* %new206 to i32*
  %dim216 = getelementptr i32* %new215, i32 -4
  store i32 %tmp201, i32* %dim216
  %dim217 = getelementptr i32* %new215, i32 -3
  store i32 %tmp198, i32* %dim217
  %new218 = bitcast i32* %new215 to double*
  store double* %new218, double** %m
  store i32 1, i32* %i
  br label %while219

while219: ; preds = %merge278, %while_body
  %i281 = load i32* %i
  %a282 = load double** %a1
  %rows_res283 = call i32 @rows(double* %a282)
  %tmp284 = icmp slt i32 %i281, %rows_res283
  br i1 %tmp284, label %while_body220, label %merge285

while_body220: ; preds = %while219
  store i32 0, i32* %j2
  store i32 0, i32* %j
  br label %while221

while221: ; preds = %merge226, %while_body220
  %j269 = load i32* %j
  %a270 = load double** %a1
  %tmp271 = icmp eq double* %a270, null
  %a272 = load double** %a1
  %dim273 = bitcast double* %a272 to i32*
  %dim274 = getelementptr i32* %dim273, i32 -3
  %dim275 = load i32* %dim274
  %tmp276 = select i1 %tmp271, i32 0, i32 %dim275
  %tmp277 = icmp slt i32 %j269, %tmp276
  br i1 %tmp277, label %while_body222, label %merge278

while_body222: ; preds = %while221
  %j223 = load i32* %j
  %j1224 = load i32* %j1
  %tmp225 = icmp ne i32 %j223, %j1224
  br i1 %tmp225, label %then227, label %else266

merge226: ; preds = %else266, %then227
  %j267 = load i32* %j
  %tmp268 = add i32 %j267, 1
  store i32 %tmp268, i32* %j
  br label %while221

then227: ; preds = %while_body222
  %i228 = load i32* %i
  %tmp229 = sub i32 %i228, 1
  %j2230 = load i32* %j2
  %m231 = load double** %m
  %i232 = load i32* %i
  %j233 = load i32* %j
  %a234 = load double** %a1
  call void @checkmatrc(double* %a234, i32 %i232, i32 %j233)
  %dim235 = bitcast double* %a234 to i32*
  %dim236 = getelementptr i32* %dim235, i32 -3
  %dim237 = load i32* %dim236
  %get238 = mul i32 %i232, %dim237
  %get239 = add i32 %j233, %get238
  %get240 = getelementptr double* %a234, i32 %get239
  %get241 = load double* %get240
  %new242 = alloca i8, i32 24
  call void @memset(i8* %new242, i32 0, i32 24)
  %new243 = getelementptr i8* %new242, i8 16
  %dim244 = bitcast i8* %new243 to i32*
  %dim245 = getelementptr i32* %dim244, i32 -1
  store i32 24, i32* %dim245
  %dim246 = bitcast i8* %new243 to i32*
  %dim247 = getelementptr i32* %dim246, i32 -2
  store i32 1, i32* %dim247
  %dim248 = bitcast i8* %new243 to i32*
  %dim249 = getelementptr i32* %dim248, i32 -4
  store i32 0, i32* %dim249
  %dim250 = bitcast i8* %new243 to i32*
  %dim251 = getelementptr i32* %dim250, i32 -3
  store i32 0, i32* %dim251
  %new252 = bitcast i8* %new243 to i32*
  %dim253 = getelementptr i32* %new252, i32 -4
  store i32 1, i32* %dim253
  %dim254 = getelementptr i32* %new252, i32 -3
  store i32 1, i32* %dim254
  %new255 = bitcast i32* %new252 to double*
  %put256 = getelementptr double* %new255, i32 0
  store double %get241, double* %put256
  call void @checkmatrc(double* %m231, i32 %tmp229, i32 %j2230)
  call void @checkmatscalar(double* %new255)
  %get257 = getelementptr double* %new255, i32 0
  %get258 = load double* %get257
  %dim259 = bitcast double* %m231 to i32*
  %dim260 = getelementptr i32* %dim259, i32 -3
  %dim261 = load i32* %dim260
  %getrc = mul i32 %tmp229, %dim261
  %getrc262 = add i32 %j2230, %getrc
  %put263 = getelementptr double* %m231, i32 %getrc262
  store double %get258, double* %put263
  %j2264 = load i32* %j2
  %tmp265 = add i32 %j2264, 1
  store i32 %tmp265, i32* %j2
  br label %merge226

else266: ; preds = %while_body222
  br label %merge226

merge278: ; preds = %while221
  %i279 = load i32* %i
  %tmp280 = add i32 %i279, 1
  store i32 %tmp280, i32* %i
  br label %while219

merge285: ; preds = %while219
  %det286 = load double** %det
  %new287 = alloca i8, i32 16
  call void @memset(i8* %new287, i32 0, i32 16)
  %new288 = getelementptr i8* %new287, i8 16
  %dim289 = bitcast i8* %new288 to i32*
  %dim290 = getelementptr i32* %dim289, i32 -1
  store i32 16, i32* %dim290
  %dim291 = bitcast i8* %new288 to i32*
```

```
%dim292 = getelementptr i32* %dim291, i32 -2
store i32 0, i32* %dim292
%dim293 = bitcast i8* %new288 to i32*
%dim294 = getelementptr i32* %dim293, i32 -4
store i32 0, i32* %dim294
%dim295 = bitcast i8* %new288 to i32*
%dim296 = getelementptr i32* %dim295, i32 -3
store i32 0, i32* %dim296
%new297 = bitcast i8* %new288 to i32*
%dim298 = getelementptr i32* %new297, i32 -4
store i32 0, i32* %dim298
%dim299 = getelementptr i32* %new297, i32 -3
store i32 0, i32* %dim299
%new300 = bitcast i32* %new297 to double*
%j1301 = load i32* %j1
%float_of = sitofp i32 %j1301 to double
%tmp302 = fadd double %float_of, 2.000000e+00
%pow_res = call double @pow(double -1.000000e+00, double %tmp302)
%new303 = alloca i8, i32 24
call void @memset(i8* %new303, i32 0, i32 24)
%new304 = getelementptr i8* %new303, i8 16
%dim305 = bitcast i8* %new304 to i32*
%dim306 = getelementptr i32* %dim305, i32 -1
store i32 24, i32* %dim306
%dim307 = bitcast i8* %new304 to i32*
%dim308 = getelementptr i32* %dim307, i32 -2
store i32 1, i32* %dim308
%dim309 = bitcast i8* %new304 to i32*
%dim310 = getelementptr i32* %dim309, i32 -4
store i32 0, i32* %dim310
%dim311 = bitcast i8* %new304 to i32*
%dim312 = getelementptr i32* %dim311, i32 -3
store i32 0, i32* %dim312
%new313 = bitcast i8* %new304 to i32*
%dim314 = getelementptr i32* %new313, i32 -4
store i32 1, i32* %dim314
%dim315 = getelementptr i32* %new313, i32 -3
store i32 1, i32* %dim315
%new316 = bitcast i32* %new313 to double*
%put317 = getelementptr double* %new316, i32 0
store double %pow_res, double* %put317
%horzcat_res318 = call double* @horzcat(double* %new300, double* %new316)
%dim319 = bitcast double* %horzcat_res318 to i32*
%dim320 = getelementptr i32* %dim319, i32 -1
%dim321 = load i32* %dim320
%new322 = alloca i8, i32 %dim321
call void @memset(i8* %new322, i32 0, i32 %dim321)
%new323 = bitcast double* %horzcat_res318 to i8*
%new324 = getelementptr i8* %new323, i8 -16
%5 = call i32 @memcpy(i8* %new322, i8* %new324, i32 %dim321)
tail call void @free(i8* %new324)
%new325 = getelementptr i8* %new322, i8 16
%cp326 = bitcast i8* %new325 to double*
%vertcat_res327 = call double* @vertcat(double* %new300, double* %cp326)
%dim328 = bitcast double* %vertcat_res327 to i32*
%dim329 = getelementptr i32* %dim328, i32 -1
%dim330 = load i32* %dim329
%new331 = alloca i8, i32 %dim330
call void @memset(i8* %new331, i32 0, i32 %dim330)
%new332 = bitcast double* %vertcat_res327 to i8*
%new333 = getelementptr i8* %new332, i8 -16
```

```
%6 = call i32 @memcpy(i8* %new331, i8* %new333, i32 %dim330)
tail call void @free(i8* %new333)
%new334 = getelementptr i8* %new331, i8 16
%cp335 = bitcast i8* %new334 to double*
%j1336 = load i32* %j1
%a337 = load double** %a1
call void @checkmatrc(double* %a337, i32 0, i32 %j1336)
%dim338 = bitcast double* %a337 to i32*
%dim339 = getelementptr i32* %dim338, i32 -3
%dim340 = load i32* %dim339
%get341 = mul i32 0, %dim340
%get342 = add i32 %j1336, %get341
%get343 = getelementptr double* %a337, i32 %get342
%get344 = load double* %get343
%new345 = alloca i8, i32 24
call void @memset(i8* %new345, i32 0, i32 24)
%new346 = getelementptr i8* %new345, i8 16
%dim347 = bitcast i8* %new346 to i32*
%dim348 = getelementptr i32* %dim347, i32 -1
store i32 24, i32* %dim348
%dim349 = bitcast i8* %new346 to i32*
%dim350 = getelementptr i32* %dim349, i32 -2
store i32 1, i32* %dim350
%dim351 = bitcast i8* %new346 to i32*
%dim352 = getelementptr i32* %dim351, i32 -4
store i32 0, i32* %dim352
%dim353 = bitcast i8* %new346 to i32*
%dim354 = getelementptr i32* %dim353, i32 -3
store i32 0, i32* %dim354
%new355 = bitcast i8* %new346 to i32*
%dim356 = getelementptr i32* %new355, i32 -4
store i32 1, i32* %dim356
%dim357 = getelementptr i32* %new355, i32 -3
store i32 1, i32* %dim357
%new358 = bitcast i32* %new355 to double*
%put359 = getelementptr double* %new358, i32 0
store double %get344, double* %put359
%mmul_res360 = call double* @mmul(double* %cp335, double* %new358)
%dim361 = bitcast double* %mmul_res360 to i32*
%dim362 = getelementptr i32* %dim361, i32 -1
%dim363 = load i32* %dim362
%new364 = alloca i8, i32 %dim363
call void @memset(i8* %new364, i32 0, i32 %dim363)
%new365 = bitcast double* %mmul_res360 to i8*
%new366 = getelementptr i8* %new365, i8 -16
%7 = call i32 @memcpy(i8* %new364, i8* %new366, i32 %dim363)
tail call void @free(i8* %new366)
%new367 = getelementptr i8* %new364, i8 16
%cp368 = bitcast i8* %new367 to double*
%new369 = alloca i8, i32 16
call void @memset(i8* %new369, i32 0, i32 16)
%new370 = getelementptr i8* %new369, i8 16
%dim371 = bitcast i8* %new370 to i32*
%dim372 = getelementptr i32* %dim371, i32 -1
store i32 16, i32* %dim372
%dim373 = bitcast i8* %new370 to i32*
%dim374 = getelementptr i32* %dim373, i32 -2
store i32 0, i32* %dim374
%dim375 = bitcast i8* %new370 to i32*
%dim376 = getelementptr i32* %dim375, i32 -4
store i32 0, i32* %dim376
```

```
%dim377 = bitcast i8* %new370 to i32*
%dim378 = getelementptr i32* %dim377, i32 -3
store i32 0, i32* %dim378
%new379 = bitcast i8* %new370 to i32*
%dim380 = getelementptr i32* %new379, i32 -4
store i32 0, i32* %dim380
%dim381 = getelementptr i32* %new379, i32 -3
store i32 0, i32* %dim381
%new382 = bitcast i32* %new379 to double*
%m383 = load double** %m
%det_res = call double @det(double* %m383)
%new384 = alloca i8, i32 24
call void @memset(i8* %new384, i32 0, i32 24)
%new385 = getelementptr i8* %new384, i8 16
%dim386 = bitcast i8* %new385 to i32*
%dim387 = getelementptr i32* %dim386, i32 -1
store i32 24, i32* %dim387
%dim388 = bitcast i8* %new385 to i32*
%dim389 = getelementptr i32* %dim388, i32 -2
store i32 1, i32* %dim389
%dim390 = bitcast i8* %new385 to i32*
%dim391 = getelementptr i32* %dim390, i32 -4
store i32 0, i32* %dim391
%dim392 = bitcast i8* %new385 to i32*
%dim393 = getelementptr i32* %dim392, i32 -3
store i32 0, i32* %dim393
%new394 = bitcast i8* %new385 to i32*
%dim395 = getelementptr i32* %new394, i32 -4
store i32 1, i32* %dim395
%dim396 = getelementptr i32* %new394, i32 -3
store i32 1, i32* %dim396
%new397 = bitcast i32* %new394 to double*
%put398 = getelementptr double* %new397, i32 0
store double %det_res, double* %put398
%horzcat_res399 = call double* @horzcat(double* %new382, double* %new397)
%dim400 = bitcast double* %horzcat_res399 to i32*
%dim401 = getelementptr i32* %dim400, i32 -1
%dim402 = load i32* %dim401
%new403 = alloca i8, i32 %dim402
call void @memset(i8* %new403, i32 0, i32 %dim402)
%new404 = bitcast double* %horzcat_res399 to i8*
%new405 = getelementptr i8* %new404, i8 -16
%8 = call i32 @memcpy(i8* %new403, i8* %new405, i32 %dim402)
tail call void @free(i8* %new405)
%new406 = getelementptr i8* %new403, i8 16
%cp407 = bitcast i8* %new406 to double*
%vertcat_res408 = call double* @vertcat(double* %new382, double* %cp407)
%dim409 = bitcast double* %vertcat_res408 to i32*
%dim410 = getelementptr i32* %dim409, i32 -1
%dim411 = load i32* %dim410
%new412 = alloca i8, i32 %dim411
call void @memset(i8* %new412, i32 0, i32 %dim411)
%new413 = bitcast double* %vertcat_res408 to i8*
%new414 = getelementptr i8* %new413, i8 -16
%9 = call i32 @memcpy(i8* %new412, i8* %new414, i32 %dim411)
tail call void @free(i8* %new414)
%new415 = getelementptr i8* %new412, i8 16
%cp416 = bitcast i8* %new415 to double*
%mmul_res417 = call double* @mmul(double* %cp368, double* %cp416)
%dim418 = bitcast double* %mmul_res417 to i32*
%dim419 = getelementptr i32* %dim418, i32 -1
```

```
  %dim420 = load i32* %dim419
  %new421 = alloca i8, i32 %dim420
  call void @memset(i8* %new421, i32 0, i32 %dim420)
  %new422 = bitcast double* %mmul_res417 to i8*
  %new423 = getelementptr i8* %new422, i8 -16
  %10 = call i32 @memcpy(i8* %new421, i8* %new423, i32 %dim420)
  tail call void @free(i8* %new423)
  %new424 = getelementptr i8* %new421, i8 16
  %cp425 = bitcast i8* %new424 to double*
  %madd_res = call double* @madd(double* %det286, double* %cp425)
  %dim426 = bitcast double* %madd_res to i32*
  %dim427 = getelementptr i32* %dim426, i32 -1
  %dim428 = load i32* %dim427
  %new429 = alloca i8, i32 %dim428
  call void @memset(i8* %new429, i32 0, i32 %dim428)
  %new430 = bitcast double* %madd_res to i8*
  %new431 = getelementptr i8* %new430, i8 -16
  %11 = call i32 @memcpy(i8* %new429, i8* %new431, i32 %dim428)
  tail call void @free(i8* %new431)
  %new432 = getelementptr i8* %new429, i8 16
  %cp433 = bitcast i8* %new432 to double*
  store double* %cp433, double** %det
  %j1434 = load i32* %j1
  %tmp435 = add i32 %j1434, 1
  store i32 %tmp435, i32* %j1
  br label %while

merge445: ; preds = %while
  br label %merge27
}

define double* @mdiv(double* %y, double* %x) {
entry:
  %y1 = alloca double*
  store double* %y, double** %y1
  %x2 = alloca double*
  store double* %x, double** %x2
  %x3 = load double** %x2
  %y4 = load double** %y1
  call void @checkmatrows(double* %y4, double* %x3)
  %x5 = load double** %x2
  %mtransp_res = call double* @mtransp(double* %x5)
  %dim = bitcast double* %mtransp_res to i32*
  %dim6 = getelementptr i32* %dim, i32 -1
  %dim7 = load i32* %dim6
  %new = alloca i8, i32 %dim7
  call void @memset(i8* %new, i32 0, i32 %dim7)
  %new8 = bitcast double* %mtransp_res to i8*
  %new9 = getelementptr i8* %new8, i8 -16
  %0 = call i32 @memcpy(i8* %new, i8* %new9, i32 %dim7)
  tail call void @free(i8* %new9)
  %new10 = getelementptr i8* %new, i8 16
  %cp = bitcast i8* %new10 to double*
  %x11 = load double** %x2
  %mmul_res = call double* @mmul(double* %cp, double* %x11)
  %dim12 = bitcast double* %mmul_res to i32*
  %dim13 = getelementptr i32* %dim12, i32 -1
  %dim14 = load i32* %dim13
  %new15 = alloca i8, i32 %dim14
  call void @memset(i8* %new15, i32 0, i32 %dim14)
  %new16 = bitcast double* %mmul_res to i8*
```

```
%new17 = getelementptr i8* %new16, i8 -16
%1 = call i32 @memcpy(i8* %new15, i8* %new17, i32 %dim14)
tail call void @free(i8* %new17)
%new18 = getelementptr i8* %new15, i8 16
%cp19 = bitcast i8* %new18 to double*
%inv_res = call double* @inv(double* %cp19)
%dim20 = bitcast double* %inv_res to i32*
%dim21 = getelementptr i32* %dim20, i32 -1
%dim22 = load i32* %dim21
%new23 = alloca i8, i32 %dim22
call void @memset(i8* %new23, i32 0, i32 %dim22)
%new24 = bitcast double* %inv_res to i8*
%new25 = getelementptr i8* %new24, i8 -16
%2 = call i32 @memcpy(i8* %new23, i8* %new25, i32 %dim22)
tail call void @free(i8* %new25)
%new26 = getelementptr i8* %new23, i8 16
%cp27 = bitcast i8* %new26 to double*
%x28 = load double** %x2
%mtransp_res29 = call double* @mtransp(double* %x28)
%dim30 = bitcast double* %mtransp_res29 to i32*
%dim31 = getelementptr i32* %dim30, i32 -1
%dim32 = load i32* %dim31
%new33 = alloca i8, i32 %dim32
call void @memset(i8* %new33, i32 0, i32 %dim32)
%new34 = bitcast double* %mtransp_res29 to i8*
%new35 = getelementptr i8* %new34, i8 -16
%3 = call i32 @memcpy(i8* %new33, i8* %new35, i32 %dim32)
tail call void @free(i8* %new35)
%new36 = getelementptr i8* %new33, i8 16
%cp37 = bitcast i8* %new36 to double*
%y38 = load double** %y1
%mmul_res39 = call double* @mmul(double* %cp37, double* %y38)
%dim40 = bitcast double* %mmul_res39 to i32*
%dim41 = getelementptr i32* %dim40, i32 -1
%dim42 = load i32* %dim41
%new43 = alloca i8, i32 %dim42
call void @memset(i8* %new43, i32 0, i32 %dim42)
%new44 = bitcast double* %mmul_res39 to i8*
%new45 = getelementptr i8* %new44, i8 -16
%4 = call i32 @memcpy(i8* %new43, i8* %new45, i32 %dim42)
tail call void @free(i8* %new45)
%new46 = getelementptr i8* %new43, i8 16
%cp47 = bitcast i8* %new46 to double*
%mmul_res48 = call double* @mmul(double* %cp27, double* %cp47)
%dim49 = bitcast double* %mmul_res48 to i32*
%dim50 = getelementptr i32* %dim49, i32 -1
%dim51 = load i32* %dim50
%new52 = alloca i8, i32 %dim51
call void @memset(i8* %new52, i32 0, i32 %dim51)
%new53 = bitcast double* %mmul_res48 to i8*
%new54 = getelementptr i8* %new53, i8 -16
%5 = call i32 @memcpy(i8* %new52, i8* %new54, i32 %dim51)
tail call void @free(i8* %new54)
%new55 = getelementptr i8* %new52, i8 16
%cp56 = bitcast i8* %new55 to double*
%dim57 = bitcast double* %cp56 to i32*
%dim58 = getelementptr i32* %dim57, i32 -1
%dim59 = load i32* %dim58
%mallocsize = mul i32 %dim59, ptrtoint (i8* getelementptr (i8* null, i32 1) to i32)
%new60 = tail call i8* @malloc(i32 %mallocsize)
call void @memset(i8* %new60, i32 0, i32 %dim59)
```

```
  %new61 = bitcast double* %cp56 to i8*
  %new62 = getelementptr i8* %new61, i8 -16
  %6 = call i32 @memcpy(i8* %new60, i8* %new62, i32 %dim59)
  %new63 = getelementptr i8* %new60, i8 16
  %cp64 = bitcast i8* %new63 to double*
  ret double* %cp64
}
```

8.3.2 Sample Program II: Logistic Regression

Listing 18: Sample target code – logistic.ll

```
; ModuleID = 'MiniMat'

@_null = global [1 x i8] zeroinitializer
@str = private unnamed_addr constant [13 x i8] c"logistic.png\00"
@str1 = private unnamed_addr constant [1 x i8] zeroinitializer
@str2 = private unnamed_addr constant [19 x i8] c"set terminal xterm\00"
@str3 = private unnamed_addr constant [14 x i8] c"set multiplot\00"
@str4 = private unnamed_addr constant [31 x i8] c"estimated logistic probability\00"
@str5 = private unnamed_addr constant [2 x i8] c"x\00"
@str6 = private unnamed_addr constant [30 x i8] c"set key top left Left reverse\00"
@str7 = private unnamed_addr constant [12 x i8] c"linespoints\00"
@str8 = private unnamed_addr constant [5 x i8] c"iter\00"
@str9 = private unnamed_addr constant [2 x i8] c":\00"
@str10 = private unnamed_addr constant [15 x i8] c"recovered data\00"
@str11 = private unnamed_addr constant [7 x i8] c"points\00"
@str12 = private unnamed_addr constant [13 x i8] c"initial data\00"
@str13 = private unnamed_addr constant [19 x i8] c"set xrange [%g:%g]\00"
@str14 = private unnamed_addr constant [19 x i8] c"set yrange [%g:%g]\00"
@str15 = private unnamed_addr constant [17 x i8] c"set terminal png\00"
@str16 = private unnamed_addr constant [16 x i8] c"set output \22%s\22\00"
@str17 = private unnamed_addr constant [6 x i8] c"%255s\00"
@str18 = private unnamed_addr constant [17 x i8] c"[%d x %d float]\0A\00"
@str19 = private unnamed_addr constant [10 x i8] c"[%d int]\0A\00"
@str20 = private unnamed_addr constant [7 x i8] c"%d %d\0A\00"
@str21 = private unnamed_addr constant [4 x i8] c"%p \00"
@str22 = private unnamed_addr constant [4 x i8] c"%s \00"
@str23 = private unnamed_addr constant [7 x i8] c"%6.2f \00"
@str24 = private unnamed_addr constant [4 x i8] c"%d \00"
@str25 = private unnamed_addr constant [2 x i8] c"\0A\00"
@str26 = private unnamed_addr constant [19 x i8] c"illegal matrix uop\00"
@str27 = private unnamed_addr constant [23 x i8] c"illegal sequence binop\00"
@str28 = private unnamed_addr constant [45 x i8] c"illegal matrix dimensions for
    multiplication\00"
@str29 = private unnamed_addr constant [29 x i8] c"illegal matrix comparison op\00"
@str30 = private unnamed_addr constant [21 x i8] c"illegal matrix binop\00"
@str31 = private unnamed_addr constant [29 x i8] c"Sequences not of same length\00"
@str32 = private unnamed_addr constant [31 x i8] c"Sequence cannot be zero length\00"
@str33 = private unnamed_addr constant [21 x i8] c"matrix is not square\00"
@str34 = private unnamed_addr constant [22 x i8] c"sequence not a scalar\00"
@str35 = private unnamed_addr constant [20 x i8] c"matrix not a scalar\00"
@str36 = private unnamed_addr constant [40 x i8] c"Matrices cannot have different
    capacity\00"
@str37 = private unnamed_addr constant [40 x i8] c"Matrices cannot have different row
    size\00"
@str38 = private unnamed_addr constant [40 x i8] c"Matrices cannot have different col
    size\00"
@str39 = private unnamed_addr constant [29 x i8] c"Matrix cannot be zero length\00"
```

```
@str40 = private unnamed_addr constant [29 x i8] c"sequence index out of bounds\00"
@str41 = private unnamed_addr constant [38 x i8] c"matrix row-column index out of
    bounds\00"
@str42 = private unnamed_addr constant [17 x i8] c"%s. Exiting...\0A\00"
@str43 = private unnamed_addr constant [34 x i8] c"matrix linear index out of bounds\00"
@str44 = private unnamed_addr constant [3 x i8] c"%f\00"
@str45 = private unnamed_addr constant [3 x i8] c"%d\00"

declare i32 @printf(i8*, ...)

declare i32 @snprintf(i8*, ...)

declare void @memset(i8*, i32, i32)

declare i32 @memcpy(i8*, i8*, i32)

declare void @gnuplot_set_ylabel(i64*, i8*)

declare void @gnuplot_set_xlabel(i64*, i8*)

declare void @gnuplot_resetplot(i64*)

declare void @gnuplot_setstyle(i64*, i8*)

declare void @gnuplot_plot_xy(i64*, double*, double*, i32, i8*)

declare void @gnuplot_close(i64*)

declare void @gnuplot_plot_equation(i64*, i8*, i8*)

declare void @gnuplot_cmd(i64*, i8*)

declare i64* @gnuplot_init()

declare i32 @scanf(i8*, i8*)

declare double @pow(double, double)

declare double @log(double)

declare double @exp(double)

declare double @fabs(double)

declare i32 @strcmp(i8*, i8*)

declare void @exit(i32)

declare double @atof(i8*)

declare i32 @atoi(i8*)

define i32 @main() {
entry:
  %x = alloca double*
  store double* null, double** %x
  %y = alloca double*
  store double* null, double** %y
  %new = alloca i8, i32 16
  call void @memset(i8* %new, i32 0, i32 16)
  %new1 = getelementptr i8* %new, i8 16
```

```
%dim = bitcast i8* %new1 to i32*
%dim2 = getelementptr i32* %dim, i32 -1
store i32 16, i32* %dim2
%dim3 = bitcast i8* %new1 to i32*
%dim4 = getelementptr i32* %dim3, i32 -2
store i32 0, i32* %dim4
%dim5 = bitcast i8* %new1 to i32*
%dim6 = getelementptr i32* %dim5, i32 -4
store i32 0, i32* %dim6
%dim7 = bitcast i8* %new1 to i32*
%dim8 = getelementptr i32* %dim7, i32 -3
store i32 0, i32* %dim8
%new9 = bitcast i8* %new1 to i32*
%dim10 = getelementptr i32* %new9, i32 -4
store i32 0, i32* %dim10
%dim11 = getelementptr i32* %new9, i32 -3
store i32 0, i32* %dim11
%new12 = bitcast i32* %new9 to double*
%stride_res = call i32* @stride(i32 2, i32 1, i32 11)
%dim13 = getelementptr i32* %stride_res, i32 -1
%dim14 = load i32* %dim13
%new15 = alloca i8, i32 %dim14
call void @memset(i8* %new15, i32 0, i32 %dim14)
%new16 = bitcast i32* %stride_res to i8*
%new17 = getelementptr i8* %new16, i8 -16
%0 = call i32 @memcpy(i8* %new15, i8* %new17, i32 %dim14)
tail call void @free(i8* %new17)
%new18 = getelementptr i8* %new15, i8 16
%cp = bitcast i8* %new18 to i32*
%mat_of_seq_res = call double* @mat_of_seq(i32* %cp)
%dim19 = bitcast double* %mat_of_seq_res to i32*
%dim20 = getelementptr i32* %dim19, i32 -1
%dim21 = load i32* %dim20
%new22 = alloca i8, i32 %dim21
call void @memset(i8* %new22, i32 0, i32 %dim21)
%new23 = bitcast double* %mat_of_seq_res to i8*
%new24 = getelementptr i8* %new23, i8 -16
%1 = call i32 @memcpy(i8* %new22, i8* %new24, i32 %dim21)
tail call void @free(i8* %new24)
%new25 = getelementptr i8* %new22, i8 16
%cp26 = bitcast i8* %new25 to double*
%stride_res27 = call i32* @stride(i32 -8, i32 1, i32 1)
%dim28 = getelementptr i32* %stride_res27, i32 -1
%dim29 = load i32* %dim28
%new30 = alloca i8, i32 %dim29
call void @memset(i8* %new30, i32 0, i32 %dim29)
%new31 = bitcast i32* %stride_res27 to i8*
%new32 = getelementptr i8* %new31, i8 -16
%2 = call i32 @memcpy(i8* %new30, i8* %new32, i32 %dim29)
tail call void @free(i8* %new32)
%new33 = getelementptr i8* %new30, i8 16
%cp34 = bitcast i8* %new33 to i32*
%mat_of_seq_res35 = call double* @mat_of_seq(i32* %cp34)
%dim36 = bitcast double* %mat_of_seq_res35 to i32*
%dim37 = getelementptr i32* %dim36, i32 -1
%dim38 = load i32* %dim37
%new39 = alloca i8, i32 %dim38
call void @memset(i8* %new39, i32 0, i32 %dim38)
%new40 = bitcast double* %mat_of_seq_res35 to i8*
%new41 = getelementptr i8* %new40, i8 -16
%3 = call i32 @memcpy(i8* %new39, i8* %new41, i32 %dim38)
```

```
tail call void @free(i8* %new41)
%new42 = getelementptr i8* %new39, i8 16
%cp43 = bitcast i8* %new42 to double*
%horzcat_res = call double* @horzcat(double* %new12, double* %cp43)
%dim44 = bitcast double* %horzcat_res to i32*
%dim45 = getelementptr i32* %dim44, i32 -1
%dim46 = load i32* %dim45
%new47 = alloca i8, i32 %dim46
call void @memset(i8* %new47, i32 0, i32 %dim46)
%new48 = bitcast double* %horzcat_res to i8*
%new49 = getelementptr i8* %new48, i8 -16
%4 = call i32 @memcpy(i8* %new47, i8* %new49, i32 %dim46)
tail call void @free(i8* %new49)
%new50 = getelementptr i8* %new47, i8 16
%cp51 = bitcast i8* %new50 to double*
%horzcat_res52 = call double* @horzcat(double* %cp51, double* %cp26)
%dim53 = bitcast double* %horzcat_res52 to i32*
%dim54 = getelementptr i32* %dim53, i32 -1
%dim55 = load i32* %dim54
%new56 = alloca i8, i32 %dim55
call void @memset(i8* %new56, i32 0, i32 %dim55)
%new57 = bitcast double* %horzcat_res52 to i8*
%new58 = getelementptr i8* %new57, i8 -16
%5 = call i32 @memcpy(i8* %new56, i8* %new58, i32 %dim55)
tail call void @free(i8* %new58)
%new59 = getelementptr i8* %new56, i8 16
%cp60 = bitcast i8* %new59 to double*
%vertcat_res = call double* @vertcat(double* %new12, double* %cp60)
%dim61 = bitcast double* %vertcat_res to i32*
%dim62 = getelementptr i32* %dim61, i32 -1
%dim63 = load i32* %dim62
%new64 = alloca i8, i32 %dim63
call void @memset(i8* %new64, i32 0, i32 %dim63)
%new65 = bitcast double* %vertcat_res to i8*
%new66 = getelementptr i8* %new65, i8 -16
%6 = call i32 @memcpy(i8* %new64, i8* %new66, i32 %dim63)
tail call void @free(i8* %new66)
%new67 = getelementptr i8* %new64, i8 16
%cp68 = bitcast i8* %new67 to double*
%mtransp_res = call double* @mtransp(double* %cp68)
%dim69 = bitcast double* %mtransp_res to i32*
%dim70 = getelementptr i32* %dim69, i32 -1
%dim71 = load i32* %dim70
%new72 = alloca i8, i32 %dim71
call void @memset(i8* %new72, i32 0, i32 %dim71)
%new73 = bitcast double* %mtransp_res to i8*
%new74 = getelementptr i8* %new73, i8 -16
%7 = call i32 @memcpy(i8* %new72, i8* %new74, i32 %dim71)
tail call void @free(i8* %new74)
%new75 = getelementptr i8* %new72, i8 16
%cp76 = bitcast i8* %new75 to double*
store double* %cp76, double** %x
%new77 = alloca i8, i32 16
call void @memset(i8* %new77, i32 0, i32 16)
%new78 = getelementptr i8* %new77, i8 16
%dim79 = bitcast i8* %new78 to i32*
%dim80 = getelementptr i32* %dim79, i32 -1
store i32 16, i32* %dim80
%dim81 = bitcast i8* %new78 to i32*
%dim82 = getelementptr i32* %dim81, i32 -2
store i32 0, i32* %dim82
```

```
%dim83 = bitcast i8* %new78 to i32*
%dim84 = getelementptr i32* %dim83, i32 -4
store i32 0, i32* %dim84
%dim85 = bitcast i8* %new78 to i32*
%dim86 = getelementptr i32* %dim85, i32 -3
store i32 0, i32* %dim86
%new87 = bitcast i8* %new78 to i32*
%dim88 = getelementptr i32* %new87, i32 -4
store i32 0, i32* %dim88
%dim89 = getelementptr i32* %new87, i32 -3
store i32 0, i32* %dim89
%new90 = bitcast i32* %new87 to double*
%ones_res = call double* @ones(i32 9, i32 1)
%dim91 = bitcast double* %ones_res to i32*
%dim92 = getelementptr i32* %dim91, i32 -1
%dim93 = load i32* %dim92
%new94 = alloca i8, i32 %dim93
call void @memset(i8* %new94, i32 0, i32 %dim93)
%new95 = bitcast double* %ones_res to i8*
%new96 = getelementptr i8* %new95, i8 -16
%8 = call i32 @memcpy(i8* %new94, i8* %new96, i32 %dim93)
tail call void @free(i8* %new96)
%new97 = getelementptr i8* %new94, i8 16
%cp98 = bitcast i8* %new97 to double*
%horzcat_res99 = call double* @horzcat(double* %new90, double* %cp98)
%dim100 = bitcast double* %horzcat_res99 to i32*
%dim101 = getelementptr i32* %dim100, i32 -1
%dim102 = load i32* %dim101
%new103 = alloca i8, i32 %dim102
call void @memset(i8* %new103, i32 0, i32 %dim102)
%new104 = bitcast double* %horzcat_res99 to i8*
%new105 = getelementptr i8* %new104, i8 -16
%9 = call i32 @memcpy(i8* %new103, i8* %new105, i32 %dim102)
tail call void @free(i8* %new105)
%new106 = getelementptr i8* %new103, i8 16
%cp107 = bitcast i8* %new106 to double*
%new108 = alloca i8, i32 24
call void @memset(i8* %new108, i32 0, i32 24)
%new109 = getelementptr i8* %new108, i8 16
%dim110 = bitcast i8* %new109 to i32*
%dim111 = getelementptr i32* %dim110, i32 -1
store i32 24, i32* %dim111
%dim112 = bitcast i8* %new109 to i32*
%dim113 = getelementptr i32* %dim112, i32 -2
store i32 1, i32* %dim113
%dim114 = bitcast i8* %new109 to i32*
%dim115 = getelementptr i32* %dim114, i32 -4
store i32 0, i32* %dim115
%dim116 = bitcast i8* %new109 to i32*
%dim117 = getelementptr i32* %dim116, i32 -3
store i32 0, i32* %dim117
%new118 = bitcast i8* %new109 to i32*
%dim119 = getelementptr i32* %new118, i32 -4
store i32 1, i32* %dim119
%dim120 = getelementptr i32* %new118, i32 -3
store i32 1, i32* %dim120
%new121 = bitcast i32* %new118 to double*
%put = getelementptr double* %new121, i32 0
store double 0.000000e+00, double* %put
%horzcat_res122 = call double* @horzcat(double* %new90, double* %new121)
%dim123 = bitcast double* %horzcat_res122 to i32*
```

```
  %dim124 = getelementptr i32* %dim123, i32 -1
  %dim125 = load i32* %dim124
  %new126 = alloca i8, i32 %dim125
  call void @memset(i8* %new126, i32 0, i32 %dim125)
  %new127 = bitcast double* %horzcat_res122 to i8*
  %new128 = getelementptr i8* %new127, i8 -16
  %10 = call i32 @memcpy(i8* %new126, i8* %new128, i32 %dim125)
  tail call void @free(i8* %new128)
  %new129 = getelementptr i8* %new126, i8 16
  %cp130 = bitcast i8* %new129 to double*
  %new131 = alloca i8, i32 24
  call void @memset(i8* %new131, i32 0, i32 24)
  %new132 = getelementptr i8* %new131, i8 16
  %dim133 = bitcast i8* %new132 to i32*
  %dim134 = getelementptr i32* %dim133, i32 -1
  store i32 24, i32* %dim134
  %dim135 = bitcast i8* %new132 to i32*
  %dim136 = getelementptr i32* %dim135, i32 -2
  store i32 1, i32* %dim136
  %dim137 = bitcast i8* %new132 to i32*
  %dim138 = getelementptr i32* %dim137, i32 -4
  store i32 0, i32* %dim138
  %dim139 = bitcast i8* %new132 to i32*
  %dim140 = getelementptr i32* %dim139, i32 -3
  store i32 0, i32* %dim140
  %new141 = bitcast i8* %new132 to i32*
  %dim142 = getelementptr i32* %new141, i32 -4
  store i32 1, i32* %dim142
  %dim143 = getelementptr i32* %new141, i32 -3
  store i32 1, i32* %dim143
  %new144 = bitcast i32* %new141 to double*
  %put145 = getelementptr double* %new144, i32 0
  store double 1.000000e+00, double* %put145
  %horzcat_res146 = call double* @horzcat(double* %new90, double* %new144)
  %dim147 = bitcast double* %horzcat_res146 to i32*
  %dim148 = getelementptr i32* %dim147, i32 -1
  %dim149 = load i32* %dim148
  %new150 = alloca i8, i32 %dim149
  call void @memset(i8* %new150, i32 0, i32 %dim149)
  %new151 = bitcast double* %horzcat_res146 to i8*
  %new152 = getelementptr i8* %new151, i8 -16
  %11 = call i32 @memcpy(i8* %new150, i8* %new152, i32 %dim149)
  tail call void @free(i8* %new152)
  %new153 = getelementptr i8* %new150, i8 16
  %cp154 = bitcast i8* %new153 to double*
  %new155 = alloca i8, i32 88
  call void @memset(i8* %new155, i32 0, i32 88)
  %new156 = getelementptr i8* %new155, i8 16
  %dim157 = bitcast i8* %new156 to i32*
  %dim158 = getelementptr i32* %dim157, i32 -1
  store i32 88, i32* %dim158
  %dim159 = bitcast i8* %new156 to i32*
  %dim160 = getelementptr i32* %dim159, i32 -2
  store i32 9, i32* %dim160
  %dim161 = bitcast i8* %new156 to i32*
  %dim162 = getelementptr i32* %dim161, i32 -4
  store i32 0, i32* %dim162
  %dim163 = bitcast i8* %new156 to i32*
  %dim164 = getelementptr i32* %dim163, i32 -3
  store i32 0, i32* %dim164
  %new165 = bitcast i8* %new156 to i32*
```

```
%dim166 = getelementptr i32* %new165, i32 -4
store i32 9, i32* %dim166
%dim167 = getelementptr i32* %new165, i32 -3
store i32 1, i32* %dim167
%new168 = bitcast i32* %new165 to double*
%horzcat_res169 = call double* @horzcat(double* %new90, double* %new168)
%dim170 = bitcast double* %horzcat_res169 to i32*
%dim171 = getelementptr i32* %dim170, i32 -1
%dim172 = load i32* %dim171
%new173 = alloca i8, i32 %dim172
call void @memset(i8* %new173, i32 0, i32 %dim172)
%new174 = bitcast double* %horzcat_res169 to i8*
%new175 = getelementptr i8* %new174, i8 -16
%12 = call i32 @memcpy(i8* %new173, i8* %new175, i32 %dim172)
tail call void @free(i8* %new175)
%new176 = getelementptr i8* %new173, i8 16
%cp177 = bitcast i8* %new176 to double*
%vertcat_res178 = call double* @vertcat(double* %new90, double* %cp177)
%dim179 = bitcast double* %vertcat_res178 to i32*
%dim180 = getelementptr i32* %dim179, i32 -1
%dim181 = load i32* %dim180
%new182 = alloca i8, i32 %dim181
call void @memset(i8* %new182, i32 0, i32 %dim181)
%new183 = bitcast double* %vertcat_res178 to i8*
%new184 = getelementptr i8* %new183, i8 -16
%13 = call i32 @memcpy(i8* %new182, i8* %new184, i32 %dim181)
tail call void @free(i8* %new184)
%new185 = getelementptr i8* %new182, i8 16
%cp186 = bitcast i8* %new185 to double*
%vertcat_res187 = call double* @vertcat(double* %cp186, double* %cp154)
%dim188 = bitcast double* %vertcat_res187 to i32*
%dim189 = getelementptr i32* %dim188, i32 -1
%dim190 = load i32* %dim189
%new191 = alloca i8, i32 %dim190
call void @memset(i8* %new191, i32 0, i32 %dim190)
%new192 = bitcast double* %vertcat_res187 to i8*
%new193 = getelementptr i8* %new192, i8 -16
%14 = call i32 @memcpy(i8* %new191, i8* %new193, i32 %dim190)
tail call void @free(i8* %new193)
%new194 = getelementptr i8* %new191, i8 16
%cp195 = bitcast i8* %new194 to double*
%vertcat_res196 = call double* @vertcat(double* %cp195, double* %cp130)
%dim197 = bitcast double* %vertcat_res196 to i32*
%dim198 = getelementptr i32* %dim197, i32 -1
%dim199 = load i32* %dim198
%new200 = alloca i8, i32 %dim199
call void @memset(i8* %new200, i32 0, i32 %dim199)
%new201 = bitcast double* %vertcat_res196 to i8*
%new202 = getelementptr i8* %new201, i8 -16
%15 = call i32 @memcpy(i8* %new200, i8* %new202, i32 %dim199)
tail call void @free(i8* %new202)
%new203 = getelementptr i8* %new200, i8 16
%cp204 = bitcast i8* %new203 to double*
%vertcat_res205 = call double* @vertcat(double* %cp204, double* %cp107)
%dim206 = bitcast double* %vertcat_res205 to i32*
%dim207 = getelementptr i32* %dim206, i32 -1
%dim208 = load i32* %dim207
%new209 = alloca i8, i32 %dim208
call void @memset(i8* %new209, i32 0, i32 %dim208)
%new210 = bitcast double* %vertcat_res205 to i8*
%new211 = getelementptr i8* %new210, i8 -16
```

```
  %16 = call i32 @memcpy(i8* %new209, i8* %new211, i32 %dim208)
  tail call void @free(i8* %new211)
  %new212 = getelementptr i8* %new209, i8 16
  %cp213 = bitcast i8* %new212 to double*
  store double* %cp213, double** %y
  %x214 = load double** %x
  %y215 = load double** %y
  %logistic_res = call double* @logistic(double* %y215, double* %x214, i8* getelementptr
      inbounds ([13 x i8]* @str, i32 0, i32 0))
  %dim216 = bitcast double* %logistic_res to i32*
  %dim217 = getelementptr i32* %dim216, i32 -1
  %dim218 = load i32* %dim217
  %new219 = alloca i8, i32 %dim218
  call void @memset(i8* %new219, i32 0, i32 %dim218)
  %new220 = bitcast double* %logistic_res to i8*
  %new221 = getelementptr i8* %new220, i8 -16
  %17 = call i32 @memcpy(i8* %new219, i8* %new221, i32 %dim218)
  tail call void @free(i8* %new221)
  %new222 = getelementptr i8* %new219, i8 16
  %cp223 = bitcast i8* %new222 to double*
  call void @printmat(double* %cp223)
  ret i32 0
}

define double* @logistic(double* %labels, double* %vars, i8* %s) {
entry:
  %labels1 = alloca double*
  store double* %labels, double** %labels1
  %vars2 = alloca double*
  store double* %vars, double** %vars2
  %s3 = alloca i8*
  store i8* %s, i8** %s3
  %iter = alloca i32
  store i32 0, i32* %iter
  %delta = alloca double
  store double 0.000000e+00, double* %delta
  %beta = alloca double*
  store double* null, double** %beta
  %prev = alloca double*
  store double* null, double** %prev
  %X = alloca double*
  store double* null, double** %X
  %z = alloca double*
  store double* null, double** %z
  %p = alloca double*
  store double* null, double** %p
  %w = alloca double*
  store double* null, double** %w
  %u = alloca double*
  store double* null, double** %u
  %x = alloca double*
  store double* null, double** %x
  %g = alloca i64*
  store i64* null, i64** %g
  %gnuplot_init_res = call i64* @gnuplot_init()
  store i64* %gnuplot_init_res, i64** %g
  %s4 = load i8** %s3
  %stringeq_res = call i1 @stringeq(i8* %s4, i8* getelementptr inbounds ([1 x i8]*
      @str1, i32 0, i32 0))
  br i1 %stringeq_res, label %then, label %else
```

```
merge: ; preds = %else, %then
  %g8 = load i64** %g
  call void @gnuplot_cmd(i64* %g8, i8* getelementptr inbounds ([14 x i8]* @str3, i32 0,
      i32 0))
  %labels9 = load double** %labels1
  %g10 = load i64** %g
  call void @gnuplot_set_yrange(i64* %g10, double* %labels9)
  %vars11 = load double** %vars2
  %g12 = load i64** %g
  call void @gnuplot_set_xrange(i64* %g12, double* %vars11)
  %g13 = load i64** %g
  call void @gnuplot_set_ylabel(i64* %g13, i8* getelementptr inbounds ([31 x i8]* @str4,
      i32 0, i32 0))
  %g14 = load i64** %g
  call void @gnuplot_set_xlabel(i64* %g14, i8* getelementptr inbounds ([2 x i8]* @str5,
      i32 0, i32 0))
  %g15 = load i64** %g
  call void @gnuplot_cmd(i64* %g15, i8* getelementptr inbounds ([30 x i8]* @str6, i32 0,
      i32 0))
  %g16 = load i64** %g
  call void @gnuplot_setstyle(i64* %g16, i8* getelementptr inbounds ([12 x i8]* @str7,
      i32 0, i32 0))
  %new = alloca i8, i32 16
  call void @memset(i8* %new, i32 0, i32 16)
  %new17 = getelementptr i8* %new, i8 16
  %dim = bitcast i8* %new17 to i32*
  %dim18 = getelementptr i32* %dim, i32 -1
  store i32 16, i32* %dim18
  %dim19 = bitcast i8* %new17 to i32*
  %dim20 = getelementptr i32* %dim19, i32 -2
  store i32 0, i32* %dim20
  %dim21 = bitcast i8* %new17 to i32*
  %dim22 = getelementptr i32* %dim21, i32 -4
  store i32 0, i32* %dim22
  %dim23 = bitcast i8* %new17 to i32*
  %dim24 = getelementptr i32* %dim23, i32 -3
  store i32 0, i32* %dim24
  %new25 = bitcast i8* %new17 to i32*
  %dim26 = getelementptr i32* %new25, i32 -4
  store i32 0, i32* %dim26
  %dim27 = getelementptr i32* %new25, i32 -3
  store i32 0, i32* %dim27
  %new28 = bitcast i32* %new25 to double*
  %vars29 = load double** %vars2
  %vars30 = load double** %vars2
  %rows_res = call i32 @rows(double* %vars30)
  %ones_res = call double* @ones(i32 %rows_res, i32 1)
  %dim31 = bitcast double* %ones_res to i32*
  %dim32 = getelementptr i32* %dim31, i32 -1
  %dim33 = load i32* %dim32
  %new34 = alloca i8, i32 %dim33
  call void @memset(i8* %new34, i32 0, i32 %dim33)
  %new35 = bitcast double* %ones_res to i8*
  %new36 = getelementptr i8* %new35, i8 -16
  %0 = call i32 @memcpy(i8* %new34, i8* %new36, i32 %dim33)
  tail call void @free(i8* %new36)
  %new37 = getelementptr i8* %new34, i8 16
  %cp = bitcast i8* %new37 to double*
  %horzcat_res = call double* @horzcat(double* %new28, double* %cp)
  %dim38 = bitcast double* %horzcat_res to i32*
  %dim39 = getelementptr i32* %dim38, i32 -1
```

```
%dim40 = load i32* %dim39
%new41 = alloca i8, i32 %dim40
call void @memset(i8* %new41, i32 0, i32 %dim40)
%new42 = bitcast double* %horzcat_res to i8*
%new43 = getelementptr i8* %new42, i8 -16
%1 = call i32 @memcpy(i8* %new41, i8* %new43, i32 %dim40)
tail call void @free(i8* %new43)
%new44 = getelementptr i8* %new41, i8 16
%cp45 = bitcast i8* %new44 to double*
%horzcat_res46 = call double* @horzcat(double* %cp45, double* %vars29)
%dim47 = bitcast double* %horzcat_res46 to i32*
%dim48 = getelementptr i32* %dim47, i32 -1
%dim49 = load i32* %dim48
%new50 = alloca i8, i32 %dim49
call void @memset(i8* %new50, i32 0, i32 %dim49)
%new51 = bitcast double* %horzcat_res46 to i8*
%new52 = getelementptr i8* %new51, i8 -16
%2 = call i32 @memcpy(i8* %new50, i8* %new52, i32 %dim49)
tail call void @free(i8* %new52)
%new53 = getelementptr i8* %new50, i8 16
%cp54 = bitcast i8* %new53 to double*
%vertcat_res = call double* @vertcat(double* %new28, double* %cp54)
%dim55 = bitcast double* %vertcat_res to i32*
%dim56 = getelementptr i32* %dim55, i32 -1
%dim57 = load i32* %dim56
%new58 = alloca i8, i32 %dim57
call void @memset(i8* %new58, i32 0, i32 %dim57)
%new59 = bitcast double* %vertcat_res to i8*
%new60 = getelementptr i8* %new59, i8 -16
%3 = call i32 @memcpy(i8* %new58, i8* %new60, i32 %dim57)
tail call void @free(i8* %new60)
%new61 = getelementptr i8* %new58, i8 16
%cp62 = bitcast i8* %new61 to double*
store double* %cp62, double** %X
%X63 = load double** %X
%tmp = icmp eq double* %X63, null
%X64 = load double** %X
%dim65 = bitcast double* %X64 to i32*
%dim66 = getelementptr i32* %dim65, i32 -3
%dim67 = load i32* %dim66
%tmp68 = select i1 %tmp, i32 0, i32 %dim67
%new69 = mul i32 %tmp68, 1
%new70 = mul i32 %new69, 8
%new71 = add i32 %new70, 16
%new72 = alloca i8, i32 %new71
call void @memset(i8* %new72, i32 0, i32 %new71)
%new73 = getelementptr i8* %new72, i8 16
%dim74 = bitcast i8* %new73 to i32*
%dim75 = getelementptr i32* %dim74, i32 -1
store i32 %new71, i32* %dim75
%dim76 = bitcast i8* %new73 to i32*
%dim77 = getelementptr i32* %dim76, i32 -2
store i32 %new69, i32* %dim77
%dim78 = bitcast i8* %new73 to i32*
%dim79 = getelementptr i32* %dim78, i32 -4
store i32 0, i32* %dim79
%dim80 = bitcast i8* %new73 to i32*
%dim81 = getelementptr i32* %dim80, i32 -3
store i32 0, i32* %dim81
%new82 = bitcast i8* %new73 to i32*
%dim83 = getelementptr i32* %new82, i32 -4
```

```
store i32 %tmp68, i32* %dim83
%dim84 = getelementptr i32* %new82, i32 -3
store i32 1, i32* %dim84
%new85 = bitcast i32* %new82 to double*
store double* %new85, double** %beta
%beta86 = load double** %beta
%new87 = alloca i8, i32 16
call void @memset(i8* %new87, i32 0, i32 16)
%new88 = getelementptr i8* %new87, i8 16
%dim89 = bitcast i8* %new88 to i32*
%dim90 = getelementptr i32* %dim89, i32 -1
store i32 16, i32* %dim90
%dim91 = bitcast i8* %new88 to i32*
%dim92 = getelementptr i32* %dim91, i32 -2
store i32 0, i32* %dim92
%dim93 = bitcast i8* %new88 to i32*
%dim94 = getelementptr i32* %dim93, i32 -4
store i32 0, i32* %dim94
%dim95 = bitcast i8* %new88 to i32*
%dim96 = getelementptr i32* %dim95, i32 -3
store i32 0, i32* %dim96
%new97 = bitcast i8* %new88 to i32*
%dim98 = getelementptr i32* %new97, i32 -4
store i32 0, i32* %dim98
%dim99 = getelementptr i32* %new97, i32 -3
store i32 0, i32* %dim99
%new100 = bitcast i32* %new97 to double*
%labels101 = load double** %labels1
%mean_res = call double @mean(double* %labels101)
%labels102 = load double** %labels1
%mean_res103 = call double @mean(double* %labels102)
%tmp104 = fsub double 1.000000e+00, %mean_res103
%tmp105 = fdiv double %mean_res, %tmp104
%log_res = call double @log(double %tmp105)
%new106 = alloca i8, i32 24
call void @memset(i8* %new106, i32 0, i32 24)
%new107 = getelementptr i8* %new106, i8 16
%dim108 = bitcast i8* %new107 to i32*
%dim109 = getelementptr i32* %dim108, i32 -1
store i32 24, i32* %dim109
%dim110 = bitcast i8* %new107 to i32*
%dim111 = getelementptr i32* %dim110, i32 -2
store i32 1, i32* %dim111
%dim112 = bitcast i8* %new107 to i32*
%dim113 = getelementptr i32* %dim112, i32 -4
store i32 0, i32* %dim113
%dim114 = bitcast i8* %new107 to i32*
%dim115 = getelementptr i32* %dim114, i32 -3
store i32 0, i32* %dim115
%new116 = bitcast i8* %new107 to i32*
%dim117 = getelementptr i32* %new116, i32 -4
store i32 1, i32* %dim117
%dim118 = getelementptr i32* %new116, i32 -3
store i32 1, i32* %dim118
%new119 = bitcast i32* %new116 to double*
%put = getelementptr double* %new119, i32 0
store double %log_res, double* %put
%horzcat_res120 = call double* @horzcat(double* %new100, double* %new119)
%dim121 = bitcast double* %horzcat_res120 to i32*
%dim122 = getelementptr i32* %dim121, i32 -1
%dim123 = load i32* %dim122
```

```
  %new124 = alloca i8, i32 %dim123
  call void @memset(i8* %new124, i32 0, i32 %dim123)
  %new125 = bitcast double* %horzcat_res120 to i8*
  %new126 = getelementptr i8* %new125, i8 -16
  %4 = call i32 @memcpy(i8* %new124, i8* %new126, i32 %dim123)
  tail call void @free(i8* %new126)
  %new127 = getelementptr i8* %new124, i8 16
  %cp128 = bitcast i8* %new127 to double*
  %vertcat_res129 = call double* @vertcat(double* %new100, double* %cp128)
  %dim130 = bitcast double* %vertcat_res129 to i32*
  %dim131 = getelementptr i32* %dim130, i32 -1
  %dim132 = load i32* %dim131
  %new133 = alloca i8, i32 %dim132
  call void @memset(i8* %new133, i32 0, i32 %dim132)
  %new134 = bitcast double* %vertcat_res129 to i8*
  %new135 = getelementptr i8* %new134, i8 -16
  %5 = call i32 @memcpy(i8* %new133, i8* %new135, i32 %dim132)
  tail call void @free(i8* %new135)
  %new136 = getelementptr i8* %new133, i8 16
  %cp137 = bitcast i8* %new136 to double*
  call void @checkmatrc(double* %beta86, i32 0, i32 0)
  call void @checkmatscalar(double* %cp137)
  %get = getelementptr double* %cp137, i32 0
  %get138 = load double* %get
  %dim139 = bitcast double* %beta86 to i32*
  %dim140 = getelementptr i32* %dim139, i32 -3
  %dim141 = load i32* %dim140
  %getrc = mul i32 0, %dim141
  %getrc142 = add i32 0, %getrc
  %put143 = getelementptr double* %beta86, i32 %getrc142
  store double %get138, double* %put143
  store double 1.000000e-06, double* %delta
  store i32 1, i32* %iter
  br label %while

then: ; preds = %entry
  %g5 = load i64** %g
  call void @gnuplot_cmd(i64* %g5, i8* getelementptr inbounds ([19 x i8]* @str2, i32 0,
      i32 0))
  br label %merge

else: ; preds = %entry
  %s6 = load i8** %s3
  %g7 = load i64** %g
  call void @gnuplot_set_png(i64* %g7, i8* %s6)
  br label %merge

while: ; preds = %while_body, %merge
  %iter596 = load i32* %iter
  %tmp597 = icmp sle i32 %iter596, 8
  %delta598 = load double* %delta
  %tmp599 = fcmp uge double %delta598, 1.000000e-06
  %tmp600 = and i1 %tmp597, %tmp599
  br i1 %tmp600, label %while_body, label %merge601

while_body: ; preds = %while
  %beta144 = load double** %beta
  %dim145 = bitcast double* %beta144 to i32*
  %dim146 = getelementptr i32* %dim145, i32 -1
  %dim147 = load i32* %dim146
  %new148 = alloca i8, i32 %dim147
```

```
call void @memset(i8* %new148, i32 0, i32 %dim147)
%new149 = bitcast double* %beta144 to i8*
%new150 = getelementptr i8* %new149, i8 -16
%6 = call i32 @memcpy(i8* %new148, i8* %new150, i32 %dim147)
%new151 = getelementptr i8* %new148, i8 16
%cp152 = bitcast i8* %new151 to double*
store double* %cp152, double** %prev
%X153 = load double** %X
%beta154 = load double** %beta
%mmul_res = call double* @mmul(double* %X153, double* %beta154)
%dim155 = bitcast double* %mmul_res to i32*
%dim156 = getelementptr i32* %dim155, i32 -1
%dim157 = load i32* %dim156
%new158 = alloca i8, i32 %dim157
call void @memset(i8* %new158, i32 0, i32 %dim157)
%new159 = bitcast double* %mmul_res to i8*
%new160 = getelementptr i8* %new159, i8 -16
%7 = call i32 @memcpy(i8* %new158, i8* %new160, i32 %dim157)
tail call void @free(i8* %new160)
%new161 = getelementptr i8* %new158, i8 16
%cp162 = bitcast i8* %new161 to double*
store double* %cp162, double** %z
%new163 = alloca i8, i32 16
call void @memset(i8* %new163, i32 0, i32 16)
%new164 = getelementptr i8* %new163, i8 16
%dim165 = bitcast i8* %new164 to i32*
%dim166 = getelementptr i32* %dim165, i32 -1
store i32 16, i32* %dim166
%dim167 = bitcast i8* %new164 to i32*
%dim168 = getelementptr i32* %dim167, i32 -2
store i32 0, i32* %dim168
%dim169 = bitcast i8* %new164 to i32*
%dim170 = getelementptr i32* %dim169, i32 -4
store i32 0, i32* %dim170
%dim171 = bitcast i8* %new164 to i32*
%dim172 = getelementptr i32* %dim171, i32 -3
store i32 0, i32* %dim172
%new173 = bitcast i8* %new164 to i32*
%dim174 = getelementptr i32* %new173, i32 -4
store i32 0, i32* %dim174
%dim175 = getelementptr i32* %new173, i32 -3
store i32 0, i32* %dim175
%new176 = bitcast i32* %new173 to double*
%new177 = alloca i8, i32 24
call void @memset(i8* %new177, i32 0, i32 24)
%new178 = getelementptr i8* %new177, i8 16
%dim179 = bitcast i8* %new178 to i32*
%dim180 = getelementptr i32* %dim179, i32 -1
store i32 24, i32* %dim180
%dim181 = bitcast i8* %new178 to i32*
%dim182 = getelementptr i32* %dim181, i32 -2
store i32 1, i32* %dim182
%dim183 = bitcast i8* %new178 to i32*
%dim184 = getelementptr i32* %dim183, i32 -4
store i32 0, i32* %dim184
%dim185 = bitcast i8* %new178 to i32*
%dim186 = getelementptr i32* %dim185, i32 -3
store i32 0, i32* %dim186
%new187 = bitcast i8* %new178 to i32*
%dim188 = getelementptr i32* %new187, i32 -4
store i32 1, i32* %dim188
```

```
%dim189 = getelementptr i32* %new187, i32 -3
store i32 1, i32* %dim189
%new190 = bitcast i32* %new187 to double*
%put191 = getelementptr double* %new190, i32 0
store double 1.000000e+00, double* %put191
%horzcat_res192 = call double* @horzcat(double* %new176, double* %new190)
%dim193 = bitcast double* %horzcat_res192 to i32*
%dim194 = getelementptr i32* %dim193, i32 -1
%dim195 = load i32* %dim194
%new196 = alloca i8, i32 %dim195
call void @memset(i8* %new196, i32 0, i32 %dim195)
%new197 = bitcast double* %horzcat_res192 to i8*
%new198 = getelementptr i8* %new197, i8 -16
%8 = call i32 @memcpy(i8* %new196, i8* %new198, i32 %dim195)
tail call void @free(i8* %new198)
%new199 = getelementptr i8* %new196, i8 16
%cp200 = bitcast i8* %new199 to double*
%vertcat_res201 = call double* @vertcat(double* %new176, double* %cp200)
%dim202 = bitcast double* %vertcat_res201 to i32*
%dim203 = getelementptr i32* %dim202, i32 -1
%dim204 = load i32* %dim203
%new205 = alloca i8, i32 %dim204
call void @memset(i8* %new205, i32 0, i32 %dim204)
%new206 = bitcast double* %vertcat_res201 to i8*
%new207 = getelementptr i8* %new206, i8 -16
%9 = call i32 @memcpy(i8* %new205, i8* %new207, i32 %dim204)
tail call void @free(i8* %new207)
%new208 = getelementptr i8* %new205, i8 16
%cp209 = bitcast i8* %new208 to double*
%new210 = alloca i8, i32 16
call void @memset(i8* %new210, i32 0, i32 16)
%new211 = getelementptr i8* %new210, i8 16
%dim212 = bitcast i8* %new211 to i32*
%dim213 = getelementptr i32* %dim212, i32 -1
store i32 16, i32* %dim213
%dim214 = bitcast i8* %new211 to i32*
%dim215 = getelementptr i32* %dim214, i32 -2
store i32 0, i32* %dim215
%dim216 = bitcast i8* %new211 to i32*
%dim217 = getelementptr i32* %dim216, i32 -4
store i32 0, i32* %dim217
%dim218 = bitcast i8* %new211 to i32*
%dim219 = getelementptr i32* %dim218, i32 -3
store i32 0, i32* %dim219
%new220 = bitcast i8* %new211 to i32*
%dim221 = getelementptr i32* %new220, i32 -4
store i32 0, i32* %dim221
%dim222 = getelementptr i32* %new220, i32 -3
store i32 0, i32* %dim222
%new223 = bitcast i32* %new220 to double*
%new224 = alloca i8, i32 24
call void @memset(i8* %new224, i32 0, i32 24)
%new225 = getelementptr i8* %new224, i8 16
%dim226 = bitcast i8* %new225 to i32*
%dim227 = getelementptr i32* %dim226, i32 -1
store i32 24, i32* %dim227
%dim228 = bitcast i8* %new225 to i32*
%dim229 = getelementptr i32* %dim228, i32 -2
store i32 1, i32* %dim229
%dim230 = bitcast i8* %new225 to i32*
%dim231 = getelementptr i32* %dim230, i32 -4
```

```
store i32 0, i32* %dim231
%dim232 = bitcast i8* %new225 to i32*
%dim233 = getelementptr i32* %dim232, i32 -3
store i32 0, i32* %dim233
%new234 = bitcast i8* %new225 to i32*
%dim235 = getelementptr i32* %new234, i32 -4
store i32 1, i32* %dim235
%dim236 = getelementptr i32* %new234, i32 -3
store i32 1, i32* %dim236
%new237 = bitcast i32* %new234 to double*
%put238 = getelementptr double* %new237, i32 0
store double 1.000000e+00, double* %put238
%horzcat_res239 = call double* @horzcat(double* %new223, double* %new237)
%dim240 = bitcast double* %horzcat_res239 to i32*
%dim241 = getelementptr i32* %dim240, i32 -1
%dim242 = load i32* %dim241
%new243 = alloca i8, i32 %dim242
call void @memset(i8* %new243, i32 0, i32 %dim242)
%new244 = bitcast double* %horzcat_res239 to i8*
%new245 = getelementptr i8* %new244, i8 -16
%10 = call i32 @memcpy(i8* %new243, i8* %new245, i32 %dim242)
tail call void @free(i8* %new245)
%new246 = getelementptr i8* %new243, i8 16
%cp247 = bitcast i8* %new246 to double*
%vertcat_res248 = call double* @vertcat(double* %new223, double* %cp247)
%dim249 = bitcast double* %vertcat_res248 to i32*
%dim250 = getelementptr i32* %dim249, i32 -1
%dim251 = load i32* %dim250
%new252 = alloca i8, i32 %dim251
call void @memset(i8* %new252, i32 0, i32 %dim251)
%new253 = bitcast double* %vertcat_res248 to i8*
%new254 = getelementptr i8* %new253, i8 -16
%11 = call i32 @memcpy(i8* %new252, i8* %new254, i32 %dim251)
tail call void @free(i8* %new254)
%new255 = getelementptr i8* %new252, i8 16
%cp256 = bitcast i8* %new255 to double*
%z257 = load double** %z
%mneg_res = call double* @mneg(double* %z257)
%dim258 = bitcast double* %mneg_res to i32*
%dim259 = getelementptr i32* %dim258, i32 -1
%dim260 = load i32* %dim259
%new261 = alloca i8, i32 %dim260
call void @memset(i8* %new261, i32 0, i32 %dim260)
%new262 = bitcast double* %mneg_res to i8*
%new263 = getelementptr i8* %new262, i8 -16
%12 = call i32 @memcpy(i8* %new261, i8* %new263, i32 %dim260)
tail call void @free(i8* %new263)
%new264 = getelementptr i8* %new261, i8 16
%cp265 = bitcast i8* %new264 to double*
%mexp_res = call double* @mexp(double* %cp265)
%dim266 = bitcast double* %mexp_res to i32*
%dim267 = getelementptr i32* %dim266, i32 -1
%dim268 = load i32* %dim267
%new269 = alloca i8, i32 %dim268
call void @memset(i8* %new269, i32 0, i32 %dim268)
%new270 = bitcast double* %mexp_res to i8*
%new271 = getelementptr i8* %new270, i8 -16
%13 = call i32 @memcpy(i8* %new269, i8* %new271, i32 %dim268)
tail call void @free(i8* %new271)
%new272 = getelementptr i8* %new269, i8 16
%cp273 = bitcast i8* %new272 to double*
```

```
%madd_res = call double* @madd(double* %cp256, double* %cp273)
%dim274 = bitcast double* %madd_res to i32*
%dim275 = getelementptr i32* %dim274, i32 -1
%dim276 = load i32* %dim275
%new277 = alloca i8, i32 %dim276
call void @memset(i8* %new277, i32 0, i32 %dim276)
%new278 = bitcast double* %madd_res to i8*
%new279 = getelementptr i8* %new278, i8 -16
%14 = call i32 @memcpy(i8* %new277, i8* %new279, i32 %dim276)
tail call void @free(i8* %new279)
%new280 = getelementptr i8* %new277, i8 16
%cp281 = bitcast i8* %new280 to double*
%mdotdiv_res = call double* @mdotdiv(double* %cp209, double* %cp281)
%dim282 = bitcast double* %mdotdiv_res to i32*
%dim283 = getelementptr i32* %dim282, i32 -1
%dim284 = load i32* %dim283
%new285 = alloca i8, i32 %dim284
call void @memset(i8* %new285, i32 0, i32 %dim284)
%new286 = bitcast double* %mdotdiv_res to i8*
%new287 = getelementptr i8* %new286, i8 -16
%15 = call i32 @memcpy(i8* %new285, i8* %new287, i32 %dim284)
tail call void @free(i8* %new287)
%new288 = getelementptr i8* %new285, i8 16
%cp289 = bitcast i8* %new288 to double*
store double* %cp289, double** %p
%p290 = load double** %p
%new291 = alloca i8, i32 16
call void @memset(i8* %new291, i32 0, i32 16)
%new292 = getelementptr i8* %new291, i8 16
%dim293 = bitcast i8* %new292 to i32*
%dim294 = getelementptr i32* %dim293, i32 -1
store i32 16, i32* %dim294
%dim295 = bitcast i8* %new292 to i32*
%dim296 = getelementptr i32* %dim295, i32 -2
store i32 0, i32* %dim296
%dim297 = bitcast i8* %new292 to i32*
%dim298 = getelementptr i32* %dim297, i32 -4
store i32 0, i32* %dim298
%dim299 = bitcast i8* %new292 to i32*
%dim300 = getelementptr i32* %dim299, i32 -3
store i32 0, i32* %dim300
%new301 = bitcast i8* %new292 to i32*
%dim302 = getelementptr i32* %new301, i32 -4
store i32 0, i32* %dim302
%dim303 = getelementptr i32* %new301, i32 -3
store i32 0, i32* %dim303
%new304 = bitcast i32* %new301 to double*
%new305 = alloca i8, i32 24
call void @memset(i8* %new305, i32 0, i32 24)
%new306 = getelementptr i8* %new305, i8 16
%dim307 = bitcast i8* %new306 to i32*
%dim308 = getelementptr i32* %dim307, i32 -1
store i32 24, i32* %dim308
%dim309 = bitcast i8* %new306 to i32*
%dim310 = getelementptr i32* %dim309, i32 -2
store i32 1, i32* %dim310
%dim311 = bitcast i8* %new306 to i32*
%dim312 = getelementptr i32* %dim311, i32 -4
store i32 0, i32* %dim312
%dim313 = bitcast i8* %new306 to i32*
%dim314 = getelementptr i32* %dim313, i32 -3
```

```
store i32 0, i32* %dim314
%new315 = bitcast i8* %new306 to i32*
%dim316 = getelementptr i32* %new315, i32 -4
store i32 1, i32* %dim316
%dim317 = getelementptr i32* %new315, i32 -3
store i32 1, i32* %dim317
%new318 = bitcast i32* %new315 to double*
%put319 = getelementptr double* %new318, i32 0
store double 1.000000e+00, double* %put319
%horzcat_res320 = call double* @horzcat(double* %new304, double* %new318)
%dim321 = bitcast double* %horzcat_res320 to i32*
%dim322 = getelementptr i32* %dim321, i32 -1
%dim323 = load i32* %dim322
%new324 = alloca i8, i32 %dim323
call void @memset(i8* %new324, i32 0, i32 %dim323)
%new325 = bitcast double* %horzcat_res320 to i8*
%new326 = getelementptr i8* %new325, i8 -16
%16 = call i32 @memcpy(i8* %new324, i8* %new326, i32 %dim323)
tail call void @free(i8* %new326)
%new327 = getelementptr i8* %new324, i8 16
%cp328 = bitcast i8* %new327 to double*
%vertcat_res329 = call double* @vertcat(double* %new304, double* %cp328)
%dim330 = bitcast double* %vertcat_res329 to i32*
%dim331 = getelementptr i32* %dim330, i32 -1
%dim332 = load i32* %dim331
%new333 = alloca i8, i32 %dim332
call void @memset(i8* %new333, i32 0, i32 %dim332)
%new334 = bitcast double* %vertcat_res329 to i8*
%new335 = getelementptr i8* %new334, i8 -16
%17 = call i32 @memcpy(i8* %new333, i8* %new335, i32 %dim332)
tail call void @free(i8* %new335)
%new336 = getelementptr i8* %new333, i8 16
%cp337 = bitcast i8* %new336 to double*
%p338 = load double** %p
%msub_res = call double* @msub(double* %cp337, double* %p338)
%dim339 = bitcast double* %msub_res to i32*
%dim340 = getelementptr i32* %dim339, i32 -1
%dim341 = load i32* %dim340
%new342 = alloca i8, i32 %dim341
call void @memset(i8* %new342, i32 0, i32 %dim341)
%new343 = bitcast double* %msub_res to i8*
%new344 = getelementptr i8* %new343, i8 -16
%18 = call i32 @memcpy(i8* %new342, i8* %new344, i32 %dim341)
tail call void @free(i8* %new344)
%new345 = getelementptr i8* %new342, i8 16
%cp346 = bitcast i8* %new345 to double*
%mdotmul_res = call double* @mdotmul(double* %p290, double* %cp346)
%dim347 = bitcast double* %mdotmul_res to i32*
%dim348 = getelementptr i32* %dim347, i32 -1
%dim349 = load i32* %dim348
%new350 = alloca i8, i32 %dim349
call void @memset(i8* %new350, i32 0, i32 %dim349)
%new351 = bitcast double* %mdotmul_res to i8*
%new352 = getelementptr i8* %new351, i8 -16
%19 = call i32 @memcpy(i8* %new350, i8* %new352, i32 %dim349)
tail call void @free(i8* %new352)
%new353 = getelementptr i8* %new350, i8 16
%cp354 = bitcast i8* %new353 to double*
store double* %cp354, double** %w
%z355 = load double** %z
%labels356 = load double** %labels1
```

```
%p357 = load double** %p
%msub_res358 = call double* @msub(double* %labels356, double* %p357)
%dim359 = bitcast double* %msub_res358 to i32*
%dim360 = getelementptr i32* %dim359, i32 -1
%dim361 = load i32* %dim360
%new362 = alloca i8, i32 %dim361
call void @memset(i8* %new362, i32 0, i32 %dim361)
%new363 = bitcast double* %msub_res358 to i8*
%new364 = getelementptr i8* %new363, i8 -16
%20 = call i32 @memcpy(i8* %new362, i8* %new364, i32 %dim361)
tail call void @free(i8* %new364)
%new365 = getelementptr i8* %new362, i8 16
%cp366 = bitcast i8* %new365 to double*
%w367 = load double** %w
%mdotdiv_res368 = call double* @mdotdiv(double* %cp366, double* %w367)
%dim369 = bitcast double* %mdotdiv_res368 to i32*
%dim370 = getelementptr i32* %dim369, i32 -1
%dim371 = load i32* %dim370
%new372 = alloca i8, i32 %dim371
call void @memset(i8* %new372, i32 0, i32 %dim371)
%new373 = bitcast double* %mdotdiv_res368 to i8*
%new374 = getelementptr i8* %new373, i8 -16
%21 = call i32 @memcpy(i8* %new372, i8* %new374, i32 %dim371)
tail call void @free(i8* %new374)
%new375 = getelementptr i8* %new372, i8 16
%cp376 = bitcast i8* %new375 to double*
%madd_res377 = call double* @madd(double* %z355, double* %cp376)
%dim378 = bitcast double* %madd_res377 to i32*
%dim379 = getelementptr i32* %dim378, i32 -1
%dim380 = load i32* %dim379
%new381 = alloca i8, i32 %dim380
call void @memset(i8* %new381, i32 0, i32 %dim380)
%new382 = bitcast double* %madd_res377 to i8*
%new383 = getelementptr i8* %new382, i8 -16
%22 = call i32 @memcpy(i8* %new381, i8* %new383, i32 %dim380)
tail call void @free(i8* %new383)
%new384 = getelementptr i8* %new381, i8 16
%cp385 = bitcast i8* %new384 to double*
store double* %cp385, double** %u
%w386 = load double** %w
%new387 = alloca i8, i32 16
call void @memset(i8* %new387, i32 0, i32 16)
%new388 = getelementptr i8* %new387, i8 16
%dim389 = bitcast i8* %new388 to i32*
%dim390 = getelementptr i32* %dim389, i32 -1
store i32 16, i32* %dim390
%dim391 = bitcast i8* %new388 to i32*
%dim392 = getelementptr i32* %dim391, i32 -2
store i32 0, i32* %dim392
%dim393 = bitcast i8* %new388 to i32*
%dim394 = getelementptr i32* %dim393, i32 -4
store i32 0, i32* %dim394
%dim395 = bitcast i8* %new388 to i32*
%dim396 = getelementptr i32* %dim395, i32 -3
store i32 0, i32* %dim396
%new397 = bitcast i8* %new388 to i32*
%dim398 = getelementptr i32* %new397, i32 -4
store i32 0, i32* %dim398
%dim399 = getelementptr i32* %new397, i32 -3
store i32 0, i32* %dim399
%new400 = bitcast i32* %new397 to double*
```

```
%new401 = alloca i8, i32 24
call void @memset(i8* %new401, i32 0, i32 24)
%new402 = getelementptr i8* %new401, i8 16
%dim403 = bitcast i8* %new402 to i32*
%dim404 = getelementptr i32* %dim403, i32 -1
store i32 24, i32* %dim404
%dim405 = bitcast i8* %new402 to i32*
%dim406 = getelementptr i32* %dim405, i32 -2
store i32 1, i32* %dim406
%dim407 = bitcast i8* %new402 to i32*
%dim408 = getelementptr i32* %dim407, i32 -4
store i32 0, i32* %dim408
%dim409 = bitcast i8* %new402 to i32*
%dim410 = getelementptr i32* %dim409, i32 -3
store i32 0, i32* %dim410
%new411 = bitcast i8* %new402 to i32*
%dim412 = getelementptr i32* %new411, i32 -4
store i32 1, i32* %dim412
%dim413 = getelementptr i32* %new411, i32 -3
store i32 1, i32* %dim413
%new414 = bitcast i32* %new411 to double*
%put415 = getelementptr double* %new414, i32 0
store double 5.000000e-01, double* %put415
%horzcat_res416 = call double* @horzcat(double* %new400, double* %new414)
%dim417 = bitcast double* %horzcat_res416 to i32*
%dim418 = getelementptr i32* %dim417, i32 -1
%dim419 = load i32* %dim418
%new420 = alloca i8, i32 %dim419
call void @memset(i8* %new420, i32 0, i32 %dim419)
%new421 = bitcast double* %horzcat_res416 to i8*
%new422 = getelementptr i8* %new421, i8 -16
%23 = call i32 @memcpy(i8* %new420, i8* %new422, i32 %dim419)
tail call void @free(i8* %new422)
%new423 = getelementptr i8* %new420, i8 16
%cp424 = bitcast i8* %new423 to double*
%vertcat_res425 = call double* @vertcat(double* %new400, double* %cp424)
%dim426 = bitcast double* %vertcat_res425 to i32*
%dim427 = getelementptr i32* %dim426, i32 -1
%dim428 = load i32* %dim427
%new429 = alloca i8, i32 %dim428
call void @memset(i8* %new429, i32 0, i32 %dim428)
%new430 = bitcast double* %vertcat_res425 to i8*
%new431 = getelementptr i8* %new430, i8 -16
%24 = call i32 @memcpy(i8* %new429, i8* %new431, i32 %dim428)
tail call void @free(i8* %new431)
%new432 = getelementptr i8* %new429, i8 16
%cp433 = bitcast i8* %new432 to double*
%mdotpow_res = call double* @mdotpow(double* %w386, double* %cp433)
%dim434 = bitcast double* %mdotpow_res to i32*
%dim435 = getelementptr i32* %dim434, i32 -1
%dim436 = load i32* %dim435
%new437 = alloca i8, i32 %dim436
call void @memset(i8* %new437, i32 0, i32 %dim436)
%new438 = bitcast double* %mdotpow_res to i8*
%new439 = getelementptr i8* %new438, i8 -16
%25 = call i32 @memcpy(i8* %new437, i8* %new439, i32 %dim436)
tail call void @free(i8* %new439)
%new440 = getelementptr i8* %new437, i8 16
%cp441 = bitcast i8* %new440 to double*
%X442 = load double** %X
%mweighted_res = call double* @mweighted(double* %X442, double* %cp441)
```

```
%dim443 = bitcast double* %mweighted_res to i32*
%dim444 = getelementptr i32* %dim443, i32 -1
%dim445 = load i32* %dim444
%new446 = alloca i8, i32 %dim445
call void @memset(i8* %new446, i32 0, i32 %dim445)
%new447 = bitcast double* %mweighted_res to i8*
%new448 = getelementptr i8* %new447, i8 -16
%26 = call i32 @memcpy(i8* %new446, i8* %new448, i32 %dim445)
tail call void @free(i8* %new448)
%new449 = getelementptr i8* %new446, i8 16
%cp450 = bitcast i8* %new449 to double*
store double* %cp450, double** %x
%w451 = load double** %w
%new452 = alloca i8, i32 16
call void @memset(i8* %new452, i32 0, i32 16)
%new453 = getelementptr i8* %new452, i8 16
%dim454 = bitcast i8* %new453 to i32*
%dim455 = getelementptr i32* %dim454, i32 -1
store i32 16, i32* %dim455
%dim456 = bitcast i8* %new453 to i32*
%dim457 = getelementptr i32* %dim456, i32 -2
store i32 0, i32* %dim457
%dim458 = bitcast i8* %new453 to i32*
%dim459 = getelementptr i32* %dim458, i32 -4
store i32 0, i32* %dim459
%dim460 = bitcast i8* %new453 to i32*
%dim461 = getelementptr i32* %dim460, i32 -3
store i32 0, i32* %dim461
%new462 = bitcast i8* %new453 to i32*
%dim463 = getelementptr i32* %new462, i32 -4
store i32 0, i32* %dim463
%dim464 = getelementptr i32* %new462, i32 -3
store i32 0, i32* %dim464
%new465 = bitcast i32* %new462 to double*
%new466 = alloca i8, i32 24
call void @memset(i8* %new466, i32 0, i32 24)
%new467 = getelementptr i8* %new466, i8 16
%dim468 = bitcast i8* %new467 to i32*
%dim469 = getelementptr i32* %dim468, i32 -1
store i32 24, i32* %dim469
%dim470 = bitcast i8* %new467 to i32*
%dim471 = getelementptr i32* %dim470, i32 -2
store i32 1, i32* %dim471
%dim472 = bitcast i8* %new467 to i32*
%dim473 = getelementptr i32* %dim472, i32 -4
store i32 0, i32* %dim473
%dim474 = bitcast i8* %new467 to i32*
%dim475 = getelementptr i32* %dim474, i32 -3
store i32 0, i32* %dim475
%new476 = bitcast i8* %new467 to i32*
%dim477 = getelementptr i32* %new476, i32 -4
store i32 1, i32* %dim477
%dim478 = getelementptr i32* %new476, i32 -3
store i32 1, i32* %dim478
%new479 = bitcast i32* %new476 to double*
%put480 = getelementptr double* %new479, i32 0
store double 5.000000e-01, double* %put480
%horzcat_res481 = call double* @horzcat(double* %new465, double* %new479)
%dim482 = bitcast double* %horzcat_res481 to i32*
%dim483 = getelementptr i32* %dim482, i32 -1
%dim484 = load i32* %dim483
```

```
  %new485 = alloca i8, i32 %dim484
  call void @memset(i8* %new485, i32 0, i32 %dim484)
  %new486 = bitcast double* %horzcat_res481 to i8*
  %new487 = getelementptr i8* %new486, i8 -16
  %27 = call i32 @memcpy(i8* %new485, i8* %new487, i32 %dim484)
  tail call void @free(i8* %new487)
  %new488 = getelementptr i8* %new485, i8 16
  %cp489 = bitcast i8* %new488 to double*
  %vertcat_res490 = call double* @vertcat(double* %new465, double* %cp489)
  %dim491 = bitcast double* %vertcat_res490 to i32*
  %dim492 = getelementptr i32* %dim491, i32 -1
  %dim493 = load i32* %dim492
  %new494 = alloca i8, i32 %dim493
  call void @memset(i8* %new494, i32 0, i32 %dim493)
  %new495 = bitcast double* %vertcat_res490 to i8*
  %new496 = getelementptr i8* %new495, i8 -16
  %28 = call i32 @memcpy(i8* %new494, i8* %new496, i32 %dim493)
  tail call void @free(i8* %new496)
  %new497 = getelementptr i8* %new494, i8 16
  %cp498 = bitcast i8* %new497 to double*
  %mdotpow_res499 = call double* @mdotpow(double* %w451, double* %cp498)
  %dim500 = bitcast double* %mdotpow_res499 to i32*
  %dim501 = getelementptr i32* %dim500, i32 -1
  %dim502 = load i32* %dim501
  %new503 = alloca i8, i32 %dim502
  call void @memset(i8* %new503, i32 0, i32 %dim502)
  %new504 = bitcast double* %mdotpow_res499 to i8*
  %new505 = getelementptr i8* %new504, i8 -16
  %29 = call i32 @memcpy(i8* %new503, i8* %new505, i32 %dim502)
  tail call void @free(i8* %new505)
  %new506 = getelementptr i8* %new503, i8 16
  %cp507 = bitcast i8* %new506 to double*
  %u508 = load double** %u
  %mweighted_res509 = call double* @mweighted(double* %u508, double* %cp507)
  %dim510 = bitcast double* %mweighted_res509 to i32*
  %dim511 = getelementptr i32* %dim510, i32 -1
  %dim512 = load i32* %dim511
  %new513 = alloca i8, i32 %dim512
  call void @memset(i8* %new513, i32 0, i32 %dim512)
  %new514 = bitcast double* %mweighted_res509 to i8*
  %new515 = getelementptr i8* %new514, i8 -16
  %30 = call i32 @memcpy(i8* %new513, i8* %new515, i32 %dim512)
  tail call void @free(i8* %new515)
  %new516 = getelementptr i8* %new513, i8 16
  %cp517 = bitcast i8* %new516 to double*
  store double* %cp517, double** %u
  %x518 = load double** %x
  %mtransp_res = call double* @mtransp(double* %x518)
  %dim519 = bitcast double* %mtransp_res to i32*
  %dim520 = getelementptr i32* %dim519, i32 -1
  %dim521 = load i32* %dim520
  %new522 = alloca i8, i32 %dim521
  call void @memset(i8* %new522, i32 0, i32 %dim521)
  %new523 = bitcast double* %mtransp_res to i8*
  %new524 = getelementptr i8* %new523, i8 -16
  %31 = call i32 @memcpy(i8* %new522, i8* %new524, i32 %dim521)
  tail call void @free(i8* %new524)
  %new525 = getelementptr i8* %new522, i8 16
  %cp526 = bitcast i8* %new525 to double*
  %x527 = load double** %x
  %mmul_res528 = call double* @mmul(double* %cp526, double* %x527)
```

```
%dim529 = bitcast double* %mmul_res528 to i32*
%dim530 = getelementptr i32* %dim529, i32 -1
%dim531 = load i32* %dim530
%new532 = alloca i8, i32 %dim531
call void @memset(i8* %new532, i32 0, i32 %dim531)
%new533 = bitcast double* %mmul_res528 to i8*
%new534 = getelementptr i8* %new533, i8 -16
%32 = call i32 @memcpy(i8* %new532, i8* %new534, i32 %dim531)
tail call void @free(i8* %new534)
%new535 = getelementptr i8* %new532, i8 16
%cp536 = bitcast i8* %new535 to double*
%inv_res = call double* @inv(double* %cp536)
%dim537 = bitcast double* %inv_res to i32*
%dim538 = getelementptr i32* %dim537, i32 -1
%dim539 = load i32* %dim538
%new540 = alloca i8, i32 %dim539
call void @memset(i8* %new540, i32 0, i32 %dim539)
%new541 = bitcast double* %inv_res to i8*
%new542 = getelementptr i8* %new541, i8 -16
%33 = call i32 @memcpy(i8* %new540, i8* %new542, i32 %dim539)
tail call void @free(i8* %new542)
%new543 = getelementptr i8* %new540, i8 16
%cp544 = bitcast i8* %new543 to double*
%x545 = load double** %x
%mtransp_res546 = call double* @mtransp(double* %x545)
%dim547 = bitcast double* %mtransp_res546 to i32*
%dim548 = getelementptr i32* %dim547, i32 -1
%dim549 = load i32* %dim548
%new550 = alloca i8, i32 %dim549
call void @memset(i8* %new550, i32 0, i32 %dim549)
%new551 = bitcast double* %mtransp_res546 to i8*
%new552 = getelementptr i8* %new551, i8 -16
%34 = call i32 @memcpy(i8* %new550, i8* %new552, i32 %dim549)
tail call void @free(i8* %new552)
%new553 = getelementptr i8* %new550, i8 16
%cp554 = bitcast i8* %new553 to double*
%u555 = load double** %u
%mmul_res556 = call double* @mmul(double* %cp554, double* %u555)
%dim557 = bitcast double* %mmul_res556 to i32*
%dim558 = getelementptr i32* %dim557, i32 -1
%dim559 = load i32* %dim558
%new560 = alloca i8, i32 %dim559
call void @memset(i8* %new560, i32 0, i32 %dim559)
%new561 = bitcast double* %mmul_res556 to i8*
%new562 = getelementptr i8* %new561, i8 -16
%35 = call i32 @memcpy(i8* %new560, i8* %new562, i32 %dim559)
tail call void @free(i8* %new562)
%new563 = getelementptr i8* %new560, i8 16
%cp564 = bitcast i8* %new563 to double*
%mmul_res565 = call double* @mmul(double* %cp544, double* %cp564)
%dim566 = bitcast double* %mmul_res565 to i32*
%dim567 = getelementptr i32* %dim566, i32 -1
%dim568 = load i32* %dim567
%new569 = alloca i8, i32 %dim568
call void @memset(i8* %new569, i32 0, i32 %dim568)
%new570 = bitcast double* %mmul_res565 to i8*
%new571 = getelementptr i8* %new570, i8 -16
%36 = call i32 @memcpy(i8* %new569, i8* %new571, i32 %dim568)
tail call void @free(i8* %new571)
%new572 = getelementptr i8* %new569, i8 16
%cp573 = bitcast i8* %new572 to double*
```

```
  store double* %cp573, double** %beta
  %beta574 = load double** %beta
  %prev575 = load double** %prev
  %msub_res576 = call double* @msub(double* %beta574, double* %prev575)
  %dim577 = bitcast double* %msub_res576 to i32*
  %dim578 = getelementptr i32* %dim577, i32 -1
  %dim579 = load i32* %dim578
  %new580 = alloca i8, i32 %dim579
  call void @memset(i8* %new580, i32 0, i32 %dim579)
  %new581 = bitcast double* %msub_res576 to i8*
  %new582 = getelementptr i8* %new581, i8 -16
  %37 = call i32 @memcpy(i8* %new580, i8* %new582, i32 %dim579)
  tail call void @free(i8* %new582)
  %new583 = getelementptr i8* %new580, i8 16
  %cp584 = bitcast i8* %new583 to double*
  %norm_res = call double @norm(double* %cp584)
  store double %norm_res, double* %delta
  call void @printstring(i8* getelementptr inbounds ([5 x i8]* @str8, i32 0, i32 0))
  %iter585 = load i32* %iter
  call void @printint(i32 %iter585)
  call void @printstring(i8* getelementptr inbounds ([2 x i8]* @str9, i32 0, i32 0))
  %delta586 = load double* %delta
  call void @printfloat(double %delta586)
  call void @println()
  %iter587 = load i32* %iter
  %string_of_int_res = call i8* @string_of_int(i32 %iter587)
  %new588 = alloca i8, i32 256
  call void @memset(i8* %new588, i32 0, i32 256)
  %38 = call i32 @memcpy(i8* %new588, i8* %string_of_int_res, i32 256)
  tail call void @free(i8* %string_of_int_res)
  %p589 = load double** %p
  %rows_res590 = call i32 @rows(double* %p589)
  %p591 = load double** %p
  %vars592 = load double** %vars2
  %g593 = load i64** %g
  call void @gnuplot_plot_xy(i64* %g593, double* %vars592, double* %p591, i32
      %rows_res590, i8* %new588)
  %iter594 = load i32* %iter
  %tmp595 = add i32 %iter594, 1
  store i32 %tmp595, i32* %iter
  br label %while

merge601: ; preds = %while
  %new602 = alloca i8, i32 16
  call void @memset(i8* %new602, i32 0, i32 16)
  %new603 = getelementptr i8* %new602, i8 16
  %dim604 = bitcast i8* %new603 to i32*
  %dim605 = getelementptr i32* %dim604, i32 -1
  store i32 16, i32* %dim605
  %dim606 = bitcast i8* %new603 to i32*
  %dim607 = getelementptr i32* %dim606, i32 -2
  store i32 0, i32* %dim607
  %dim608 = bitcast i8* %new603 to i32*
  %dim609 = getelementptr i32* %dim608, i32 -4
  store i32 0, i32* %dim609
  %dim610 = bitcast i8* %new603 to i32*
  %dim611 = getelementptr i32* %dim610, i32 -3
  store i32 0, i32* %dim611
  %new612 = bitcast i8* %new603 to i32*
  %dim613 = getelementptr i32* %new612, i32 -4
  store i32 0, i32* %dim613
```

```
%dim614 = getelementptr i32* %new612, i32 -3
store i32 0, i32* %dim614
%new615 = bitcast i32* %new612 to double*
%new616 = alloca i8, i32 24
call void @memset(i8* %new616, i32 0, i32 24)
%new617 = getelementptr i8* %new616, i8 16
%dim618 = bitcast i8* %new617 to i32*
%dim619 = getelementptr i32* %dim618, i32 -1
store i32 24, i32* %dim619
%dim620 = bitcast i8* %new617 to i32*
%dim621 = getelementptr i32* %dim620, i32 -2
store i32 1, i32* %dim621
%dim622 = bitcast i8* %new617 to i32*
%dim623 = getelementptr i32* %dim622, i32 -4
store i32 0, i32* %dim623
%dim624 = bitcast i8* %new617 to i32*
%dim625 = getelementptr i32* %dim624, i32 -3
store i32 0, i32* %dim625
%new626 = bitcast i8* %new617 to i32*
%dim627 = getelementptr i32* %new626, i32 -4
store i32 1, i32* %dim627
%dim628 = getelementptr i32* %new626, i32 -3
store i32 1, i32* %dim628
%new629 = bitcast i32* %new626 to double*
%put630 = getelementptr double* %new629, i32 0
store double 1.000000e+00, double* %put630
%horzcat_res631 = call double* @horzcat(double* %new615, double* %new629)
%dim632 = bitcast double* %horzcat_res631 to i32*
%dim633 = getelementptr i32* %dim632, i32 -1
%dim634 = load i32* %dim633
%new635 = alloca i8, i32 %dim634
call void @memset(i8* %new635, i32 0, i32 %dim634)
%new636 = bitcast double* %horzcat_res631 to i8*
%new637 = getelementptr i8* %new636, i8 -16
%39 = call i32 @memcpy(i8* %new635, i8* %new637, i32 %dim634)
tail call void @free(i8* %new637)
%new638 = getelementptr i8* %new635, i8 16
%cp639 = bitcast i8* %new638 to double*
%vertcat_res640 = call double* @vertcat(double* %new615, double* %cp639)
%dim641 = bitcast double* %vertcat_res640 to i32*
%dim642 = getelementptr i32* %dim641, i32 -1
%dim643 = load i32* %dim642
%new644 = alloca i8, i32 %dim643
call void @memset(i8* %new644, i32 0, i32 %dim643)
%new645 = bitcast double* %vertcat_res640 to i8*
%new646 = getelementptr i8* %new645, i8 -16
%40 = call i32 @memcpy(i8* %new644, i8* %new646, i32 %dim643)
tail call void @free(i8* %new646)
%new647 = getelementptr i8* %new644, i8 16
%cp648 = bitcast i8* %new647 to double*
%new649 = alloca i8, i32 16
call void @memset(i8* %new649, i32 0, i32 16)
%new650 = getelementptr i8* %new649, i8 16
%dim651 = bitcast i8* %new650 to i32*
%dim652 = getelementptr i32* %dim651, i32 -1
store i32 16, i32* %dim652
%dim653 = bitcast i8* %new650 to i32*
%dim654 = getelementptr i32* %dim653, i32 -2
store i32 0, i32* %dim654
%dim655 = bitcast i8* %new650 to i32*
%dim656 = getelementptr i32* %dim655, i32 -4
```

```
store i32 0, i32* %dim656
%dim657 = bitcast i8* %new650 to i32*
%dim658 = getelementptr i32* %dim657, i32 -3
store i32 0, i32* %dim658
%new659 = bitcast i8* %new650 to i32*
%dim660 = getelementptr i32* %new659, i32 -4
store i32 0, i32* %dim660
%dim661 = getelementptr i32* %new659, i32 -3
store i32 0, i32* %dim661
%new662 = bitcast i32* %new659 to double*
%new663 = alloca i8, i32 24
call void @memset(i8* %new663, i32 0, i32 24)
%new664 = getelementptr i8* %new663, i8 16
%dim665 = bitcast i8* %new664 to i32*
%dim666 = getelementptr i32* %dim665, i32 -1
store i32 24, i32* %dim666
%dim667 = bitcast i8* %new664 to i32*
%dim668 = getelementptr i32* %dim667, i32 -2
store i32 1, i32* %dim668
%dim669 = bitcast i8* %new664 to i32*
%dim670 = getelementptr i32* %dim669, i32 -4
store i32 0, i32* %dim670
%dim671 = bitcast i8* %new664 to i32*
%dim672 = getelementptr i32* %dim671, i32 -3
store i32 0, i32* %dim672
%new673 = bitcast i8* %new664 to i32*
%dim674 = getelementptr i32* %new673, i32 -4
store i32 1, i32* %dim674
%dim675 = getelementptr i32* %new673, i32 -3
store i32 1, i32* %dim675
%new676 = bitcast i32* %new673 to double*
%put677 = getelementptr double* %new676, i32 0
store double 1.000000e+00, double* %put677
%horzcat_res678 = call double* @horzcat(double* %new662, double* %new676)
%dim679 = bitcast double* %horzcat_res678 to i32*
%dim680 = getelementptr i32* %dim679, i32 -1
%dim681 = load i32* %dim680
%new682 = alloca i8, i32 %dim681
call void @memset(i8* %new682, i32 0, i32 %dim681)
%new683 = bitcast double* %horzcat_res678 to i8*
%new684 = getelementptr i8* %new683, i8 -16
%41 = call i32 @memcpy(i8* %new682, i8* %new684, i32 %dim681)
tail call void @free(i8* %new684)
%new685 = getelementptr i8* %new682, i8 16
%cp686 = bitcast i8* %new685 to double*
%vertcat_res687 = call double* @vertcat(double* %new662, double* %cp686)
%dim688 = bitcast double* %vertcat_res687 to i32*
%dim689 = getelementptr i32* %dim688, i32 -1
%dim690 = load i32* %dim689
%new691 = alloca i8, i32 %dim690
call void @memset(i8* %new691, i32 0, i32 %dim690)
%new692 = bitcast double* %vertcat_res687 to i8*
%new693 = getelementptr i8* %new692, i8 -16
%42 = call i32 @memcpy(i8* %new691, i8* %new693, i32 %dim690)
tail call void @free(i8* %new693)
%new694 = getelementptr i8* %new691, i8 16
%cp695 = bitcast i8* %new694 to double*
%X696 = load double** %X
%mneg_res697 = call double* @mneg(double* %X696)
%dim698 = bitcast double* %mneg_res697 to i32*
%dim699 = getelementptr i32* %dim698, i32 -1
```

```
%dim700 = load i32* %dim699
%new701 = alloca i8, i32 %dim700
call void @memset(i8* %new701, i32 0, i32 %dim700)
%new702 = bitcast double* %mneg_res697 to i8*
%new703 = getelementptr i8* %new702, i8 -16
%43 = call i32 @memcpy(i8* %new701, i8* %new703, i32 %dim700)
tail call void @free(i8* %new703)
%new704 = getelementptr i8* %new701, i8 16
%cp705 = bitcast i8* %new704 to double*
%beta706 = load double** %beta
%mmul_res707 = call double* @mmul(double* %cp705, double* %beta706)
%dim708 = bitcast double* %mmul_res707 to i32*
%dim709 = getelementptr i32* %dim708, i32 -1
%dim710 = load i32* %dim709
%new711 = alloca i8, i32 %dim710
call void @memset(i8* %new711, i32 0, i32 %dim710)
%new712 = bitcast double* %mmul_res707 to i8*
%new713 = getelementptr i8* %new712, i8 -16
%44 = call i32 @memcpy(i8* %new711, i8* %new713, i32 %dim710)
tail call void @free(i8* %new713)
%new714 = getelementptr i8* %new711, i8 16
%cp715 = bitcast i8* %new714 to double*
%mexp_res716 = call double* @mexp(double* %cp715)
%dim717 = bitcast double* %mexp_res716 to i32*
%dim718 = getelementptr i32* %dim717, i32 -1
%dim719 = load i32* %dim718
%new720 = alloca i8, i32 %dim719
call void @memset(i8* %new720, i32 0, i32 %dim719)
%new721 = bitcast double* %mexp_res716 to i8*
%new722 = getelementptr i8* %new721, i8 -16
%45 = call i32 @memcpy(i8* %new720, i8* %new722, i32 %dim719)
tail call void @free(i8* %new722)
%new723 = getelementptr i8* %new720, i8 16
%cp724 = bitcast i8* %new723 to double*
%madd_res725 = call double* @madd(double* %cp695, double* %cp724)
%dim726 = bitcast double* %madd_res725 to i32*
%dim727 = getelementptr i32* %dim726, i32 -1
%dim728 = load i32* %dim727
%new729 = alloca i8, i32 %dim728
call void @memset(i8* %new729, i32 0, i32 %dim728)
%new730 = bitcast double* %madd_res725 to i8*
%new731 = getelementptr i8* %new730, i8 -16
%46 = call i32 @memcpy(i8* %new729, i8* %new731, i32 %dim728)
tail call void @free(i8* %new731)
%new732 = getelementptr i8* %new729, i8 16
%cp733 = bitcast i8* %new732 to double*
%mdotdiv_res734 = call double* @mdotdiv(double* %cp648, double* %cp733)
%dim735 = bitcast double* %mdotdiv_res734 to i32*
%dim736 = getelementptr i32* %dim735, i32 -1
%dim737 = load i32* %dim736
%new738 = alloca i8, i32 %dim737
call void @memset(i8* %new738, i32 0, i32 %dim737)
%new739 = bitcast double* %mdotdiv_res734 to i8*
%new740 = getelementptr i8* %new739, i8 -16
%47 = call i32 @memcpy(i8* %new738, i8* %new740, i32 %dim737)
tail call void @free(i8* %new740)
%new741 = getelementptr i8* %new738, i8 16
%cp742 = bitcast i8* %new741 to double*
store double* %cp742, double** %p
%vars743 = load double** %vars2
%rows_res744 = call i32 @rows(double* %vars743)
```

```
  %p745 = load double** %p
  %vars746 = load double** %vars2
  %g747 = load i64** %g
  call void @gnuplot_plot_xy(i64* %g747, double* %vars746, double* %p745, i32
      %rows_res744, i8* getelementptr inbounds ([15 x i8]* @str10, i32 0, i32 0))
  %g748 = load i64** %g
  call void @gnuplot_setstyle(i64* %g748, i8* getelementptr inbounds ([7 x i8]* @str11,
      i32 0, i32 0))
  %vars749 = load double** %vars2
  %rows_res750 = call i32 @rows(double* %vars749)
  %labels751 = load double** %labels1
  %vars752 = load double** %vars2
  %g753 = load i64** %g
  call void @gnuplot_plot_xy(i64* %g753, double* %vars752, double* %labels751, i32
      %rows_res750, i8* getelementptr inbounds ([13 x i8]* @str12, i32 0, i32 0))
  %g754 = load i64** %g
  call void @gnuplot_close(i64* %g754)
  %beta755 = load double** %beta
  %dim756 = bitcast double* %beta755 to i32*
  %dim757 = getelementptr i32* %dim756, i32 -1
  %dim758 = load i32* %dim757
  %mallocsize = mul i32 %dim758, ptrtoint (i8* getelementptr (i8* null, i32 1) to i32)
  %new759 = tail call i8* @malloc(i32 %mallocsize)
  call void @memset(i8* %new759, i32 0, i32 %dim758)
  %new760 = bitcast double* %beta755 to i8*
  %new761 = getelementptr i8* %new760, i8 -16
  %48 = call i32 @memcpy(i8* %new759, i8* %new761, i32 %dim758)
  %new762 = getelementptr i8* %new759, i8 16
  %cp763 = bitcast i8* %new762 to double*
  ret double* %cp763
}

define double* @mweighted(double* %x, double* %w) {
entry:
  %x1 = alloca double*
  store double* %x, double** %x1
  %w2 = alloca double*
  store double* %w, double** %w2
  %y = alloca double*
  store double* null, double** %y
  %r = alloca i32
  store i32 0, i32* %r
  %w3 = load double** %w2
  %x4 = load double** %x1
  call void @checkmatrows(double* %x4, double* %w3)
  %x5 = load double** %x1
  %tmp = icmp eq double* %x5, null
  %x6 = load double** %x1
  %dim = bitcast double* %x6 to i32*
  %dim7 = getelementptr i32* %dim, i32 -3
  %dim8 = load i32* %dim7
  %tmp9 = select i1 %tmp, i32 0, i32 %dim8
  %x10 = load double** %x1
  %rows_res = call i32 @rows(double* %x10)
  %new = mul i32 %rows_res, %tmp9
  %new11 = mul i32 %new, 8
  %new12 = add i32 %new11, 16
  %new13 = alloca i8, i32 %new12
  call void @memset(i8* %new13, i32 0, i32 %new12)
  %new14 = getelementptr i8* %new13, i8 16
  %dim15 = bitcast i8* %new14 to i32*
  %dim16 = getelementptr i32* %dim15, i32 -1
  store i32 %new12, i32* %dim16
  %dim17 = bitcast i8* %new14 to i32*
  %dim18 = getelementptr i32* %dim17, i32 -2
  store i32 %new, i32* %dim18
  %dim19 = bitcast i8* %new14 to i32*
  %dim20 = getelementptr i32* %dim19, i32 -4
  store i32 0, i32* %dim20
  %dim21 = bitcast i8* %new14 to i32*
  %dim22 = getelementptr i32* %dim21, i32 -3
  store i32 0, i32* %dim22
  %new23 = bitcast i8* %new14 to i32*
  %dim24 = getelementptr i32* %new23, i32 -4
  store i32 %rows_res, i32* %dim24
  %dim25 = getelementptr i32* %new23, i32 -3
  store i32 %tmp9, i32* %dim25
  %new26 = bitcast i32* %new23 to double*
  store double* %new26, double** %y
  store i32 0, i32* %r
  br label %while

while: ; preds = %while_body, %entry
  %r133 = load i32* %r
  %x134 = load double** %x1
  %rows_res135 = call i32 @rows(double* %x134)
  %tmp136 = icmp slt i32 %r133, %rows_res135
  br i1 %tmp136, label %while_body, label %merge

while_body: ; preds = %while
  %r27 = load i32* %r
  %y28 = load double** %y
  %tmp29 = icmp eq double* %y28, null
  %y30 = load double** %y
  %dim31 = bitcast double* %y30 to i32*
  %dim32 = getelementptr i32* %dim31, i32 -3
  %dim33 = load i32* %dim32
  %tmp34 = select i1 %tmp29, i32 0, i32 %dim33
  %tmp35 = sub i32 %tmp34, 1
  %stride_res = call i32* @stride(i32 0, i32 1, i32 %tmp35)
  %dim36 = getelementptr i32* %stride_res, i32 -1
  %dim37 = load i32* %dim36
  %new38 = alloca i8, i32 %dim37
  call void @memset(i8* %new38, i32 0, i32 %dim37)
  %new39 = bitcast i32* %stride_res to i8*
  %new40 = getelementptr i8* %new39, i8 -16
  %0 = call i32 @memcpy(i8* %new38, i8* %new40, i32 %dim37)
  tail call void @free(i8* %new40)
  %new41 = getelementptr i8* %new38, i8 16
  %cp = bitcast i8* %new41 to i32*
  %y42 = load double** %y
  %r43 = load i32* %r
  %y44 = load double** %y
  %tmp45 = icmp eq double* %y44, null
  %y46 = load double** %y
  %dim47 = bitcast double* %y46 to i32*
  %dim48 = getelementptr i32* %dim47, i32 -3
  %dim49 = load i32* %dim48
  %tmp50 = select i1 %tmp45, i32 0, i32 %dim49
  %tmp51 = sub i32 %tmp50, 1
  %stride_res52 = call i32* @stride(i32 0, i32 1, i32 %tmp51)
  %dim53 = getelementptr i32* %stride_res52, i32 -1
  %dim54 = load i32* %dim53
  %new55 = alloca i8, i32 %dim54
  call void @memset(i8* %new55, i32 0, i32 %dim54)
  %new56 = bitcast i32* %stride_res52 to i8*
  %new57 = getelementptr i8* %new56, i8 -16
  %1 = call i32 @memcpy(i8* %new55, i8* %new57, i32 %dim54)
  tail call void @free(i8* %new57)
  %new58 = getelementptr i8* %new55, i8 16
  %cp59 = bitcast i8* %new58 to i32*
  %x60 = load double** %x1
  %new61 = alloca i8, i32 20
  call void @memset(i8* %new61, i32 0, i32 20)
  %new62 = getelementptr i8* %new61, i8 16
  %dim63 = bitcast i8* %new62 to i32*
  %dim64 = getelementptr i32* %dim63, i32 -1
  store i32 20, i32* %dim64
  %dim65 = bitcast i8* %new62 to i32*
  %dim66 = getelementptr i32* %dim65, i32 -2
  store i32 1, i32* %dim66
  %dim67 = bitcast i8* %new62 to i32*
  %dim68 = getelementptr i32* %dim67, i32 -4
  store i32 0, i32* %dim68
  %dim69 = bitcast i8* %new62 to i32*
  %dim70 = getelementptr i32* %dim69, i32 -3
  store i32 0, i32* %dim70
  %new71 = bitcast i8* %new62 to i32*
  %put = getelementptr i32* %new71, i32 0
  store i32 %r43, i32* %put
  %mselect_res = call double* @mselect(double* %x60, i32* %new71, i32* %cp59)
  %dim72 = bitcast double* %mselect_res to i32*
  %dim73 = getelementptr i32* %dim72, i32 -1
  %dim74 = load i32* %dim73
  %new75 = alloca i8, i32 %dim74
  call void @memset(i8* %new75, i32 0, i32 %dim74)
  %new76 = bitcast double* %mselect_res to i8*
  %new77 = getelementptr i8* %new76, i8 -16
  %2 = call i32 @memcpy(i8* %new75, i8* %new77, i32 %dim74)
  tail call void @free(i8* %new77)
  %new78 = getelementptr i8* %new75, i8 16
  %cp79 = bitcast i8* %new78 to double*
  %r80 = load i32* %r
  %w81 = load double** %w2
  call void @checkmatrc(double* %w81, i32 %r80, i32 0)
  %dim82 = bitcast double* %w81 to i32*
  %dim83 = getelementptr i32* %dim82, i32 -3
  %dim84 = load i32* %dim83
  %get = mul i32 %r80, %dim84
  %get85 = add i32 0, %get
  %get86 = getelementptr double* %w81, i32 %get85
  %get87 = load double* %get86
  %new88 = alloca i8, i32 24
  call void @memset(i8* %new88, i32 0, i32 24)
  %new89 = getelementptr i8* %new88, i8 16
  %dim90 = bitcast i8* %new89 to i32*
  %dim91 = getelementptr i32* %dim90, i32 -1
  store i32 24, i32* %dim91
  %dim92 = bitcast i8* %new89 to i32*
  %dim93 = getelementptr i32* %dim92, i32 -2
  store i32 1, i32* %dim93
  %dim94 = bitcast i8* %new89 to i32*
  %dim95 = getelementptr i32* %dim94, i32 -4
  store i32 0, i32* %dim95
  %dim96 = bitcast i8* %new89 to i32*
  %dim97 = getelementptr i32* %dim96, i32 -3
  store i32 0, i32* %dim97
  %new98 = bitcast i8* %new89 to i32*
  %dim99 = getelementptr i32* %new98, i32 -4
  store i32 1, i32* %dim99
  %dim100 = getelementptr i32* %new98, i32 -3
  store i32 1, i32* %dim100
  %new101 = bitcast i32* %new98 to double*
  %put102 = getelementptr double* %new101, i32 0
  store double %get87, double* %put102
  %mmul_res = call double* @mmul(double* %cp79, double* %new101)
  %dim103 = bitcast double* %mmul_res to i32*
  %dim104 = getelementptr i32* %dim103, i32 -1
  %dim105 = load i32* %dim104
  %new106 = alloca i8, i32 %dim105
  call void @memset(i8* %new106, i32 0, i32 %dim105)
  %new107 = bitcast double* %mmul_res to i8*
  %new108 = getelementptr i8* %new107, i8 -16
  %3 = call i32 @memcpy(i8* %new106, i8* %new108, i32 %dim105)
  tail call void @free(i8* %new108)
  %new109 = getelementptr i8* %new106, i8 16
  %cp110 = bitcast i8* %new109 to double*
  %new111 = alloca i8, i32 20
  call void @memset(i8* %new111, i32 0, i32 20)
  %new112 = getelementptr i8* %new111, i8 16
  %dim113 = bitcast i8* %new112 to i32*
  %dim114 = getelementptr i32* %dim113, i32 -1
  store i32 20, i32* %dim114
  %dim115 = bitcast i8* %new112 to i32*
  %dim116 = getelementptr i32* %dim115, i32 -2
  store i32 1, i32* %dim116
  %dim117 = bitcast i8* %new112 to i32*
  %dim118 = getelementptr i32* %dim117, i32 -4
  store i32 0, i32* %dim118
  %dim119 = bitcast i8* %new112 to i32*
  %dim120 = getelementptr i32* %dim119, i32 -3
  store i32 0, i32* %dim120
  %new121 = bitcast i8* %new112 to i32*
  %put122 = getelementptr i32* %new121, i32 0
  store i32 %r27, i32* %put122
  %massign_res = call double* @massign(double* %y42, i32* %new121, i32* %cp, double*
      %cp110)
  %dim123 = bitcast double* %massign_res to i32*
  %dim124 = getelementptr i32* %dim123, i32 -1
  %dim125 = load i32* %dim124
  %new126 = alloca i8, i32 %dim125
  call void @memset(i8* %new126, i32 0, i32 %dim125)
  %new127 = bitcast double* %massign_res to i8*
  %new128 = getelementptr i8* %new127, i8 -16
  %4 = call i32 @memcpy(i8* %new126, i8* %new128, i32 %dim125)
  tail call void @free(i8* %new128)
  %new129 = getelementptr i8* %new126, i8 16
  %cp130 = bitcast i8* %new129 to double*
  %r131 = load i32* %r
  %tmp132 = add i32 %r131, 1
  store i32 %tmp132, i32* %r
  br label %while

merge: ; preds = %while
```

```
  %y137 = load double** %y
  %dim138 = bitcast double* %y137 to i32*
  %dim139 = getelementptr i32* %dim138, i32 -1
  %dim140 = load i32* %dim139
  %mallocsize = mul i32 %dim140, ptrtoint (i8* getelementptr (i8* null, i32 1) to i32)
  %new141 = tail call i8* @malloc(i32 %mallocsize)
  call void @memset(i8* %new141, i32 0, i32 %dim140)
  %new142 = bitcast double* %y137 to i8*
  %new143 = getelementptr i8* %new142, i8 -16
  %5 = call i32 @memcpy(i8* %new141, i8* %new143, i32 %dim140)
  %new144 = getelementptr i8* %new141, i8 16
  %cp145 = bitcast i8* %new144 to double*
  ret double* %cp145
}

define void @gnuplot_set_xrange(i64* %g, double* %x) {
entry:
  %g1 = alloca i64*
  store i64* %g, i64** %g1
  %x2 = alloca double*
  store double* %x, double** %x2
  %x3 = load double** %x2
  %min_res = call double @min(double* %x3)
  %x4 = load double** %x2
  %max_res = call double @max(double* %x4)
  %new = alloca i8, i32 256
  call void @memset(i8* %new, i32 0, i32 256)
  %snpr = call i32 (i8*, ...)* @snprintf(i8* %new, i32 256, i8* getelementptr inbounds
      ([19 x i8]* @str13, i32 0, i32 0), double %min_res, double %max_res)
  %g5 = load i64** %g1
  call void @gnuplot_cmd(i64* %g5, i8* %new)
  ret void
}

define void @gnuplot_set_yrange(i64* %g, double* %y) {
entry:
  %g1 = alloca i64*
  store i64* %g, i64** %g1
  %y2 = alloca double*
  store double* %y, double** %y2
  %y3 = load double** %y2
  %min_res = call double @min(double* %y3)
  %y4 = load double** %y2
  %max_res = call double @max(double* %y4)
  %new = alloca i8, i32 256
  call void @memset(i8* %new, i32 0, i32 256)
  %snpr = call i32 (i8*, ...)* @snprintf(i8* %new, i32 256, i8* getelementptr inbounds
      ([19 x i8]* @str14, i32 0, i32 0), double %min_res, double %max_res)
  %g5 = load i64** %g1
  call void @gnuplot_cmd(i64* %g5, i8* %new)
  ret void
}

define void @gnuplot_set_png(i64* %g, i8* %f) {
entry:
  %g1 = alloca i64*
  store i64* %g, i64** %g1
  %f2 = alloca i8*
  store i8* %f, i8** %f2
  %g3 = load i64** %g1
  call void @gnuplot_cmd(i64* %g3, i8* getelementptr inbounds ([17 x i8]* @str15, i32 0,
      i32 0))
  %f4 = load i8** %f2
  %new = alloca i8, i32 256
  call void @memset(i8* %new, i32 0, i32 256)
  %snpr = call i32 (i8*, ...)* @snprintf(i8* %new, i32 256, i8* getelementptr inbounds
      ([16 x i8]* @str16, i32 0, i32 0), i8* %f4)
  %g5 = load i64** %g1
  call void @gnuplot_cmd(i64* %g5, i8* %new)
  ret void
}
```