COMS 4115:PROGRAMMING LANGUAGE AND TRANLATOR

CAML: COMPOSING AMAZING MUSIC LANGUAGE

May 20, 2019

Yipeng Zhang	Yuanji Huang	Kunjian Liao
yz3396	yh3059	kl3063

Contents

1 INTRODUCTION

CAML is a simple but powerful strongly-typed programming language that aims to facilitate digital music generation. It supports first class function, structs, arrays with a static scoped C-like syntax.

1.1 Motivations

Digital audio software has become increasingly popular in recent years. Such software frees musicians om the constraints of physical instruments and provides instant feedback for any spark of inspiration and creativity.

Many digital software features a powerful GUI interface for composer to feel like writing on a music sheet. Since most complex music contains some sort of repeated pieces, many efforts are wasted on manual editing of redundant works. Also, many software of these kinds suffer from space/memory inefficiency. For example, GarageBand takes over more than 1.7GB of storage.
Our language CAML, a simple, small, but powerful tool, is designed to minimize the amount of repetitive works and enables musicians without any coding experience to compose digital music.

1.2 Characteristic of CAML

CAML provides powerful functionality including first class functions, Array manipulations, structs to help composers to focus on the structure of their scripts and produce complicated digital composition on the fly.
By manipulating the sequences of tracks and chords, composers could synthesized multiple tracks with a full range of different instruments with a simple keyboard.

1.2.1 Simple

CAML strives to simplify the syntax for musical composers and aims to serve people with no programming experience. We employ many tweaks, known as syntactic sugar, to make things easier for CAML developers to read and to express.
For instance, CAML has a universal print function that can take string, integer, bool, or pitch

as input and print the corresponding object accordingly. Another useful feature is variable can be declared and initialized in the some line. Array manipulation such as concat, multiple useful functionality in the standard library, all of which makes the CAML programmer to write less and accomplish more.

1.2.2 Extensible

CAML enables user-defined library and custom functions and structures. As music composition could use a lot of repetitive efforts, programmers could take advantage of a self-written library or a custom structure to streamline their production. CAML is platform-independent as long as a compatible LLVM, OCaml and dependencies versions are satisfied.

1.2.3 Powerful

CAML does not limit itself to music composition only. Our goal is to be able to implement a variety of complex tasks and common algorithms such as Dijkstra's algorithm, bubble-sort algorithm, and many others efficiently. The details of implementations can be found at Appendix.

2 TUTORIAL

2.1 Environment Setup

To use this language, you need to install setup the OCaml and LLVM environment on your computer. Mac users are recommended to install OCaml through Homebrew, using the following commands:

```
brew install ocaml
brew install opam
```

The next step is to install LLVM and OCaml LLVM library. LLVM can also be easily installed through H omebrew by:

```
brew install llvm
```

You need to remember where Homebrew places LLVM executables and the version of your LLVM, because the version of the OCaml LLVM library must match the version of the LLVM.

The commands to install Ocaml LLVM Library is:

```
opam install llvm.3.6
```

You may need to install its dependencies if your command line tool request you to do so.

2.2 Installation and Compilation

The source code of CAML language and its translator is available on Github:

`https://github.com/kliao4243/Caml_Project.git`

You can download this repository as a .zip file to your local device and start compiling. First enter the source directory and run **build.sh**. The script build the standard library and CAML compiler. If build failed, delete the existed **caml.native** file and compile it again. Second enter the midifile and run another **build.sh** here. It will initialize the Midifile package and build an executable **midi-generator**, which will be used to generate songs. Then you are good to write some music.
To compile code, please use the **run.sh** to generate song or print raw output:

```
run.sh -s code.cl //generate .ll file and song
run.sh -p code.cl // generate .ll file and print raw output
run.sh -o code.cl out// generate .ll file and save raw output to out
```

The output of CAML built-in generate music function is a MIDI file. To play the music you compose, you need to download a MIDI-compatible music software. Our recommendations are Guitar Pro and GarageBand. GuitarPro is available throught the link:

`https://www.guitar-pro.com/en/index.php`

GarageBand can be downloaded on Apple App Store.

3 LANGUAGE MANUAL

3.1 Primitive Types

Boolean (bool) : A value indicating true or false, such as True.

Integer (int) : A 32-bit signed integer, such as 256.

Floating point (double) : A 128-bits sequence of number with a decimal point, such as

256.652.

String (String) : A sequence of character we defined above, whose length is variable, such as 'Even KUN Cai can compose amazing music with the help of CAML' or "Even KUN Cai can compose amazing music with the help of CAML". String supports two types of escape characters '\n' (newline) and '\t' (tab).

Pitch (Pitch): A 24-bit value denoting the pitch of a note. Notes C-D-E-F-G-A-B are represented as 1, 2, 3, 4, 5, 6, 7. Natural notes, flat notes, and sharp notes are followed by ^, b, and # respectively. The octave is represented by another followed number, and 4 represents the central octave. For example, 1^4 represents Middle C (C4), or MIDI note number 60, frequency 261.63 Hz.

3.2 Array

An `Array` is a set of objects of the same type enclosed in square brackets. Elements in an array are divided by commas. Type of an array must be specified during variable declaration. Array supports indexing and concatenation. An array looks like the following

```
int[5] int_array = [1, 2, 3, 4, 5]
int_array = int_array @ [6,7] /* int_array = [1, 2, 3, 4, 5, 6, 7]*/
pitch[4] song = [1^4, 2^4, 3^4, 4^4] /* pitch array*/
int[5] nested = [[1,2,3],[2,3],[3],[4],[5,6]]// nested array is supported
```

Like most languages, all elements in one array must be of the same data type. The following is an example of a illegal array:

```
int[6] int_array = [1, 2, 3, 4, 5, 'Amazing']
```

One thing worth noting is that assigning a pre-defined Array to a new Array doesn't create a new Array but points the second array to the same address of the same element. To create a copy of the same Array, one should use the concatenation operator with an empty Array. For more details, check section 3.8 Array Operator.

```
int[4] int_array = [1, 2, 3, 4];
int[4] new_int_array = int_array; /* this points to the same address*/
new_int_array[0] = 10; /* int_array = [10, 2, 3, 4]*/
int[4] new_int_array = int_array@[]; /* this copies the array */
```

```
new_int_array[0] = 1000; /* int_array = [10, 2, 3, 4]*/ remained unchanged
```

3.3 User Defined Type (Struct)

User-defined type is supported in CAML. Users can define a data type that is not included in the standard library by using the keyword **struct**. The name of a new struct has to start with underscore _. The member field of a struct can contain several variable s of any primitive data types. An array of primitive data type can also be a member of a struct. The following is an example of defining a new struct:

```
struct _demoNote = {
    String Artist;
    Pitch demo;
};
```

Once a struct is defined, it can be used just as other system defined data type. Variables inside a struct can be accessed by using member access operator **(.)**. The following is a legal example of creating a user defined type object and modifying this object:

```
_demoNote practice;
practice.Artist = "Hanon";
practice.demo = 4^4;
```

3.4 Track

In our definition, a song is the combination of one or multiple tracks. `Track` is a system-defined complex struct consisting of a melody array, a rhythm array, size and instrument. Pitch and rhythm array carry the information of every note in this track. The code for instrument can be found in **instrument-reference.txt**. The following is an example of a valid track:

```
_track twinkle;
twinkle.melody = {4^4, 4^4, 5^4, 5^4};
twinkle.rhythm = [1, 1, 1, 2];
twinkle.size = 4;
twinkle.instrument = 1 /*Piano*/
```

A `Track` can also be built by built-in function:

```
_track t = build_track(1,4,{4^4, 4^4, 5^4, 5^4},[1, 1, 1, 2]);
```

3.5 Variables and Assignment

CAML is a strongly-typed language. Types of variables are predefined and cannot be inferred to other types. Certain operators are allowable only with certain data types. `=` is used to assign value to an identifier:

```
int one = 1;
int two = 1+1;
String s = "hello world";
Pitch c = 1^4;
Pitch c_sharp = 1#4;
Pitch[7] star = {1^4, 1^4, 5^4, 5^4, 6^4, 6^4, 5^4};
int[4] arr = [1,2,3,4];
```

3.6 Arithmetic Operators

CAML supports arithmetic operators +, -, *, /. The unary - operator has the highest precedence, followed by the binary *, /, followed by the binary + and - operators. All arithmetic operators are left-associative. Arithmetic operators cannot be applied to different types of operands. For example,

```
int a = 1 + 1; /* valid */
int a = 2 + 0.5; /* invalid */
```

3.7 Logical Operators

Logical operators include `==`, `!=` , `<=`, `>=`, `<`, `>` and keywords `and`, `or`. `<=`, `>=`, `<`, `>` have the highest precedence, followed by `==`, `!=`, followed by `and`, `or`. In equality comparison, primitives are compared by value. Note that most logical operators don't support for `Pitch`.

```
bool test = 1 == 1; /* return True */
Pitch c_sharp = 1#4;
```

```
Pitch d_flat = 2b4;
bool a = c_sharp == d_flat; /* syntax error */
```

3.8 Array Operator

We have two array special operator: @, which stands for concatenation, and [], which is used to access an element by index. The concatenation operator takes either two pre-defined Array or a temporary Array literal and create a new Array of the same type with the combined set of elements. Note that array concatenation creates another array instead of just copying over the pointer, unlike the assign operator, which is much more expensive and should be used with caution on large arrays.

```
int[3] Arr = [1,2,3];
int[6] Arr2 = Arr @ Arr; /* Arr2 = [1,2,3,1,2,3]*/
int[6] Arr3 = Arr @ [1,2,3]; /* Arr3 = [1,2,3,1,2,3]*/
Arr3[2] = 30; /* This would not affect Arr*/
Arr[2] = Arr3[2]; /* This would affect Arr*/
```

3.9 Comments

/**/ is the comment symbol.

```
/* This is a comment */
```

3.10 Conditional Statements

$$conditional\ expr \rightarrow \textbf{if}\ condition\textbf{:}\ do\ something$$

$$[\textbf{else:}do\ something\ else]^*$$

Each expression after an if statement should be an expression that evaluates to a boolean that is either true or false. The else statement following the if is optional.
If the expression evaluates to true then the expression indented following the subsequent colon is executed. Otherwise, in the case there's an else statement, the expression indented following the colon after else is executed.

In the case of nested if statements, then the else binds to the if block with the same indentation. Parentheses around the conditional statements are optional. As the example shown below:

```
A = 3;
B = 4;
if (A > 0){ /* this is the first if statement */
    if (B > 0){ /* this is the second if statement*/
        B = B+5;}
else: {/* this else matches the first if statement*/
    B = B-5;}
    }
```

3.11 loop Statements

$$loops expr \rightarrow \textbf{while}\, condition\textbf{:}\, do something$$

$$[break/continue]^*$$

Each expression after an if statement should be an expression that evaluates to a boolean that is either true or false. If the expression evaluates to true then the expression indented following the subsequent colon is executed.
In the case of a break statement, we break out of the while loops and continue executing the program afterwards. In the case of a continue statement, we skip the current iteration and start the next iteration immediately.
In the case of nested while statements, then any break or continue binds to the while block with the preceding indentation. As the example shown below:

```
i = 0;
j = 0;
sum = 0;
while (i < 10){ /* this is the first while statement*/
    while (j < 5){ /*this is the second while statement*/
        sum = sum+j;
        if (j % 3 == 0){
            break;} /*this break matches the second while*/
    i = i+1;
```

```
    if (i == 5){
        continue; /*this continue the first while and start from i*/ = i+1}}
    sum = sum + i}
```

While loops can be very helpful in repeating the same rhyme and tracks multiple times.

3.12 Module

A program must declare the module it needs to import using **#include** keyword at its beginning. Once import a module, a program can use any functions and user defined types in that module. The following is a valid example of using a module:

```
/*This is song.cl*/
struct _song = {
    String name;
    int rating;
};
```

```
/*This is another file*/
#include "src/stdlib.cl";
#include "song.cl";

_song demoSong;
demoSong.name = "The Entertainer";
demoSong.rating = 5;
```

3.13 Function

The function is defined by a **def** keyword, and the control flow inside a function is also determined by indentation. The return type of a function must be specified before the def keyword, and the types of argument also need to be specified when defining a function. The following is a legal function definition example:

```
int def add_one(int input):
    return input+1
```

3.14 Scope

The scope of a name binding, such as a user defined type or a variable, is region of a program where the binding is valid. Such a region is referred to as a scope block. Outside the scope block of a variable, the name may refer to a different entity (it may have a different binding), or to nothing at all (it may be unbound).

In CAML, the scoping policy is natural and very similar to other prevalent programming languages such as Python and Java. A variable's scope block is basically the continuous space inside the same braces as this variable. In the example below:

```
int i=0

int print_one(){
    int i = 1;
    print(i);
}

print_one();
print(i);
```

The program prints 1 and then prints 0. The scope block of the variable i inside the print_one function is all space inside the program. The scope block of the variable i outside the print_one function is the whole program except the print_one function.

3.15 STL

Our standard library is written both in C and CAML. It provides built-in functions for covert-ing pitch, building track and generating songs. The built-in struct type **Track** is also declared in STL.

```
int pitch_to_int(pitch x);
void generate_music(_track x);
_track build_track(int instrument, int size, pitch[] melody, int[] rhythm);
int ran(int range);
```

To convert the user input pitches into readable integers for midi generator, `pitch_to_int` translates pitch to integer through this graph.

Note	Octave										
	-1	0	1	2	3	4	5	6	7	8	9
C	0	12	24	36	48	**60**	72	84	96	108	120
C#	1	13	25	37	49	**61**	73	85	97	109	121
D	2	14	26	38	50	**62**	74	86	98	110	122
D#	3	15	27	39	51	**63**	75	87	99	111	123
E	4	16	28	40	52	**64**	76	88	100	112	124
F	5	17	29	41	53	**65**	77	89	101	113	125
F#	6	18	30	42	54	**66**	78	90	102	114	126
G	7	19	31	43	55	**67**	79	91	103	115	127
G#	8	20	32	44	56	**68**	80	92	104	116	
A	9	21	33	45	57	**69**	81	93	105	117	
A#	10	22	34	46	58	**70**	82	94	106	118	
B	11	23	35	47	59	**71**	83	95	107	119	

`build_track` serves as a constructor for tracks. It takes member variables as input and return the corresponding track. The maximum size for a track is 50.
`generate_music` is used to generate our standard format output for midi generator. It should be call at the end of program. Calling this function more than once means adding more than one track into the song. The output would automatically merge tracks and come out with a song with multiple parallel tracks.

3.16 Program Structure

A CAML program is may exist entirely within a single source file, but more commonly, programs that aiming at composing will consist of the standard library, and will also link with files from existing files. For example, an sample program for twinkle twinkle little star:

```
#include "src/stdlib.cl";
int main()
{
   int i;
   int a;
   _track track;
   Pitch[7] first = [1^4,1^4,5^4,5^4,6^4,6^4,5^4];
    Pitch[7] second = [4^4,4^4,3^4,3^4,2^4,2^4,1^4];
    Pitch[14] total = first@second;
   int[] rhythm = [4,4,4,4,4,4,2,4,4,4,4,4,4,2];
```

```
    track = build_track(1, 14, total, rhythm);
    generate_music(track);
    return 0;
}
```

The compiler first loads functions and structs in the included files that declared on the top of program, then it will load the other functions or structs that defined in the current program. Compiler will try to find `main` function and execute the code in its block. Users can create whatever note they want by defining the melody and rhythm array. Note that melody array and rhythm array should have the same lengths. After finished track building users need to call `generate_music` to generate standard output for midi generator before return.

4 PROJECT PLAN

4.1 Planning and Development Process

We submitted the draft of language reference manual on March 6th and begin planning on the implementation of the whole project.

We began to have a rigorous and detailed plan after the spring break. Before the due date of the first Hello World assignment, we had several times of discussion about what programming language features we must support to enable CAML to compose real songs. Before the end of March, we determined that pitch data type, array, and struct are the most element functionalities of CAML. They are crucial to represent a song.

On April 9th, we finished the implementation of array and string print (Hello World). After that, Our team generally meet once or twice a week depending on everyone's schedule. Most of the meetings happen on Wednesday and Thursday. Our work style was to stay and work together for several hours, and solved the problems and bugs during meetings unless it was identified as very easy to tackle.

User defined type was a challenging feature to develop and took us lots of time. We finished the scanner, parser, ast, semantic check, sast in two meetings, and spent another two meetings to implement the code generation. The struct was finished on April 30th.

During the last two weeks before final demonstration, we spent significantly more time on this project. We finished the music related part of this project, including linking C standard library, generating MIDI files before May 10th. And we developed more useful built-in func-

tions, and added other functionalities in the translator like improving array concatenation before the final presentation.

4.2 Programming Style Guide

As mentioned, our team typically work together, and had very smooth communication and punctual feedback. For these reasons, we did not have very specific rules on programming style, However, we were well-coordinated to follow some unspecified rules, like add one tab of indentation after each let command in OCaml, use the same naming convention as MicroC, etc.

4.3 Project Timeline

March 31th	Hello World worked
April 7th	Add primitive data type pitch
April 9th	Array worked
April 17th	Struct can output SAST
April 19th	Integrate print function
April 21th	Funtion can take array as input
April 30th	Struct worked
May 1st	Arrays can be concatenated
May 3rd	Can import modules and standard library
May 5th	Caml standard library finished
May 6th	C standard library finished
May 10th	Can generate MIDI file
May 15th	Final version after lots of small adjustment

4.4 Roles and Responsibilities of Each member

Name	Role	Responsibilities
Yuanji Huang	System Architect, Tester	struct, variable assignment, c standard library, caml standard library, midi file generation, testing
Kunjian Liao	Manager, Tester	pitch, struct, include, caml standard library, testing
Yipeng Zhang	Language Guru	array, combine print function, array concatenation, array size function, testing

4.5 Language and Environment

We used Sublime as text editor and ran the code on Mac Terminal. We developed on OCaml for more than 90% of time. We wrote a standard library in C to support music generation, we also used C++ to generate a executable file to get access to MIDI library.

4.6 Git Log

```
9703d77 yuanjihuang yh3059@columbia.edu add up and down
2d7a8b1 yuanjihuang yh3059@columbia.edu final version
9badded wolszhang  wolszhang@gmail.com bubble sort
25678ba wolszhang  wolszhang@gmail.com warning fixed
26036f9 yuanjihuang yh3059@columbia.edu delete useless files
44792ea yuanjihuang yh3059@columbia.edu merge
1672c03 yuanjihuang yh3059@columbia.edu add stdlib
5f10958 wolszhang  wolszhang@gmail.com delete imcomplete work
b229bf2 yuanjihuang yh3059@columbia.edu add midifile
```

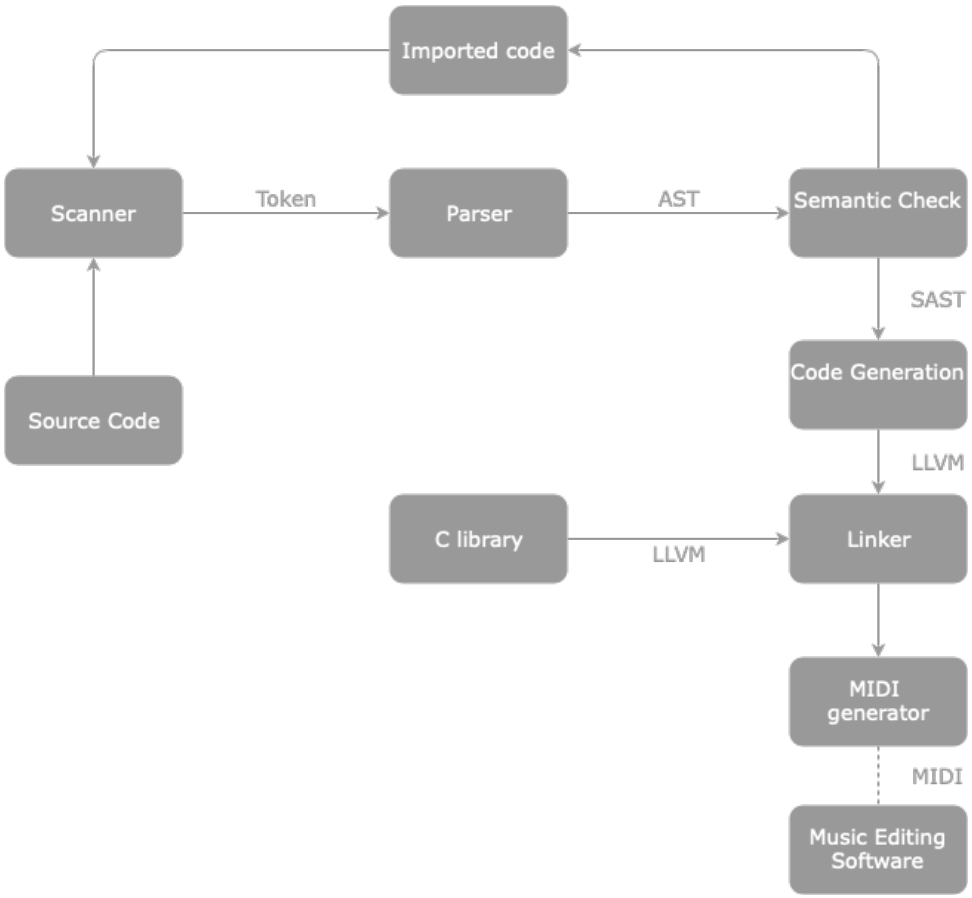
```
2a5b28d yuanjihuang yh3059@columbia.edu add c and caml stdlib
6c36e3b yuanjihuang yh3059@columbia.edu merge
b3f06a1 Liao kunjian@dyn-160-39-170-66.dyn.columbia.edu add import struct test
711a6d4 yuanjihuang yh3059@columbia.edu merged
65c0f4b kliao4243 32171833+kliao4243@users.noreply.github.com Merge pull
    request #6 from kliao4243/std_branch
b774130 Liao kunjian@dyn-160-39-170-66.dyn.columbia.edu merge parser.mly
fad24ad Liao kunjian@dyn-160-39-148-139.dyn.columbia.edu merge include
ba9103f yuanjihuang yh3059@columbia.edu before merge
d8cf479 wolszhang wolszhang@gmail.com include concat function
a249f3a Liao kunjian@dyn-160-39-148-142.dyn.columbia.edu include module finish
f9b10a8 wolszhang wolszhang@gmail.com fix p
f876408 Liao kunjian@dyn-160-39-148-142.dyn.columbia.edu try parse program
    into semantic
c5aa95c yuanjihuang yh3059@columbia.edu no change
b262c28 wolszhang wolszhang@gmail.com Array size function
085171d Liao kunjian@dyn-160-39-148-142.dyn.columbia.edu std
7c8bfbe yuanjihuang yh3059@columbia.edu function can take array as argument
c2e0e14 yuanjihuang yh3059@columbia.edu array concat
a2e72f8 yuanjihuang yh3059@columbia.edu Merge branch 'master' of
    https://github.com/kliao4243/Caml_Project
f973377 yuanjihuang yh3059@columbia.edu modify array
e19183d Liao kunjian@dyn-160-39-148-108.dyn.columbia.edu version revert back
    to version 1
6df02f5 yuanjihuang yh3059@columbia.edu merge
969da72 yuanjihuang yh3059@columbia.edu version1
d4b94c1 Liao kunjian@dyn-160-39-148-108.dyn.columbia.edu forgot to delete head
    index
c2dcefd Liao kunjian@dyn-160-39-148-108.dyn.columbia.edu merge with integrated
    print
3869560 Liao kunjian@dyn-160-39-148-108.dyn.columbia.edu code generation
    struct in progress
15c0c12 yuanjihuang yh3059@columbia.edu merge confict
5a95b15 yuanjihuang yh3059@columbia.edu delete build
3394d73 yuanjihuang yh3059@columbia.edu assign when declaring
1ef369e wolszhang wolszhang@gmail.com Merge pull request #4 from
    kliao4243/Array
```

```
8cf563c wolszhang  wolszhang@gmail.com combined print
eeb6f7d wolszhang  wolszhang@gmail.com Merge pull request #3 from
   kliao4243/Array
cf84a5a wolszhang  wolszhang@gmail.com Merge branch 'master' of
   https://github.com/kliao4243/Caml_Project into array
e2529d8 wolszhang  wolszhang@gmail.com ee
748fe1c yuanjihuang yh3059@columbia.edu handle conflict
8.7127E+09 yuanjihuang yh3059@columbia.edu merge
f84f7bf yuanjihuang yh3059@columbia.edu add array assign
d56e1ea Liao  kunjian@dyn-160-39-148-108.dyn.columbia.edu solve dot
   shift/reduce conflicts
904814a wolszhang  wolszhang@gmail.com pitch_to_int prototype
530df1d Liao  kunjian@dyn-160-39-171-219.dyn.columbia.edu merge struct except
   code generation to main branch
1b7df40 wolszhang  wolszhang@gmail.com print function updated
3c6746d Liao  kunjian@dyn-160-39-148-108.dyn.columbia.edu struct
107f982 Liao  kunjian@dyn-160-39-148-108.dyn.columbia.edu struct except code
   generation
7ec8fe1 wolszhang  wolszhang@gmail.com Merge pull request #2 from
   kliao4243/Array
e07f5e2 wolszhang  wolszhang@gmail.com fix shift/reduce conflicts
04f7c92 wolszhang  wolszhang@gmail.com Merge pull request #1 from
   kliao4243/Array
21334ac wolszhang  wolszhang@gmail.com merged version m Please enter the commit
   message for your changes. Lines starting
dc9fc88 wolszhang  wolszhang@gmail.com Merge branch 'master' of
   https://github.com/kliao4243/Caml_Project into Array
f153fab wolszhang  wolszhang@gmail.com Array function completed Yipeng
01982c8 wolszhang  wolszhang@gmail.com add array
22c7ce4 wolszhang  wolszhang@gmail.com include Access
1e06a6d Liao  kunjian@dyn-160-39-170-158.dyn.columbia.edu new pitch
9b517ae Liao  kunjian@dyn-160-39-170-158.dyn.columbia.edu pitch correction
a644129 Liao  kunjian@dyn-160-39-170-158.dyn.columbia.edu pitch
25bf953 wolszhang  wolszhang@gmail.com include size function and iterator
5026a72 wolszhang  wolszhang@gmail.com include Array support Array<type> a
9020f56 yuanjihuang yh3059@columbia.edu helloworld works
2a1e1cc yuanjihuang yh3059@columbia.edu first
```

5 ARCHITECTURAL DESIGN

5.1 System Block Diagram

CAML compiler follows a sequence of process to transform source code into a midi file, including lexical analysis, syntax analysis, semantic checking, IR generation, machine code generation and midi generation.



5.2 Lexical Analysis

In lexical analysis phase, scanner scans the source code as a stream of characters and converts it into tokens. If any illegal characters are detected, scanner will throw lexical exceptions. Note that characters in comment are not scanned. This part is done by Yipeng and Kunjian.

5.3 Syntax Analysis

The next phase is the parsing phase. It takes the token produced by lexical analysis as input and generates an abstract syntax tree. In CAML, program is a list of declarations, while declaration can be function, variable or struct declaration. One of CAML special design in parser is when Array is declared, user can choose to define the size or use default value. The size information will be carried by token `Array` and makes it easier for Array concatenation and iteration. This part is done by Yipeng and Kunjian.

5.4 Semantic Checking

In this phase, compiler will traverse the AST and generate SAST. Standard library is also added in this phase: if token `include` is found in program, compiler will try to open the included file, go through all phases before and attach the included AST before current AST. Compiler also adds declarations of build-in function that written in C into symbol table in this phase, otherwise program cannot pass semantic checking when calling build-in functions. This part is done by Yuanji, Yipeng and Kunjian.

5.5 IR generation

In this last front-end phase, the compiler generates LLVM code of the source code for the target machine. Except for the primitive types such as integer and float, types like array and struct are pointer actually. The most challenging thing here is how to use limited LLVM methods to implement Array concatenation. Our solution is to define a new pointer, fetch each element in these two arrays by for loop, and assign it to the new pointer one by one. But if someone try to concatenate multiple arrays at one time:

```
int[8] a = [1,2]@[2,3]@[3,4]@[4,5];
```

When handling the second and the following '@', the compiler has to concatenate the processed arrays again instead of using stored results. That makes multiple array concatenation a memory-consuming operation. This part is done by Yuanji and Yipeng.

5.6 Machine code generation

In this phase C standard library is linked to the program and executable file is generated. By executing this file, standard format output is stored and read by midi generator. This part is done by Yuanji.

5.7 Midi generation

CAML utilizes C++ library Midifile to generate .mid file. .mid file can be open with GarageBand or Guitar Pro. This part is done by Yuanji.

6 TEST PLAN

We have two separate test suites for our language and newly implemented features: unit test and integration test. In unit test, we make sure each piece of the codes was correctly implemented and achieved exactly what we expected. In integration test, we combined functionality to check if features work well with each other. For most or part of the functionalities, we divide our testing responsibility into the following:
Kunjian Liao: struct, include
Yuanji Huang: stdlib.c, stdlib.cl
Yipeng Zhang: array, pointer, operators
General(all): if, for, while, ifelse, function

6.1 Unit Test

The followings are the test cases and the final output (.exe output).

Test: print int

```
/* author: Yuanji Huang */
int main(){
    print(123);
}
```

```
123
```

Test: print float

```
/* author: Yuanji Huang */
int main(){
    print(123.2);
```

```
}
```

```
123.2
```

Test: print string

```
/* author: Yuanji Huang */
int main(){
    print("helloworld");
}
```

```
helloworld
```

Test: print pitch

```
/* author: Yuanji Huang */
int main(){
    print(4^4);
    print(5#4);
    print(4b4);
}
```

```
4^4
5#4
4b4
```

Test: assign int

```
/* author: Yuanji Huang */
int main(){
    int b;
    int a = 1;
    b = 2;
    print(a);
```

```
    print(b);
}
```

```
1
2
```

Test: assign float

```
/* author: Yuanji Huang */
int main(){
    float b;
    float a = 1.1;
    b = 2.23;
    print(a);
    print(b);
}
```

```
1.1
2.23
```

Test: assign string

```
/* author: Yuanji Huang */
int main(){
    String a = "helloworld";
    print(a);
}
```

```
helloworld
```

Test: assign pitch

```
/* author: Yuanji Huang */
int main(){
```

```
    Pitch a = 2b4;
    Pitch b = 3^7;
    Pitch c = 4#5;
    print(a);
    print(b);
    print(c);
}
```

```
2b4
3^7
4#5
```

Test: build track

```
/* author: Yuanji Huang */
#include "src/stdlib.cl";
int main(){
    _track t;
    t = build_track(1,2,[4b4,4#4],[4,4]);
    print(t.size);
    print(t.instrument);
    print(t.melody[0]);
    print(t.rhythm[0]);
    print(t.melody[1]);
    print(t.rhythm[1]);
}
```

```
2
1
4b4
4
4#4
4
```

Test: generate music

```
/* author: Yuanji Huang */
#include "src/stdlib.cl";
int main(){
    _track t;
    t = build_track(1,2,[4b4,4#4],[4,4]);
    generate_music(t);
}
```

```
track_start
1
2
64
4
66
4
```

Test: struct with single member

```
/*Kunjian Liao*/
#include "src/stdlib.cl";
struct _test{
    int test;
};
int main(){
   _test a;
   a.test = 1;
   print(a.test);
}
```

```
1
```

Test: struct with multiple members

```
/*Kunjian Liao*/
#include "src/stdlib.cl";
```

```
struct _test{
    int test;
    String test2;
};
int main(){
   _test a;
   a.test = 1;
   a.test2 = "Hello World";
   print(a.test);
   print(a.test2);
}
```

```
1
Hello World
```

Test: struct with pitch member

```
/*Kunjian Liao*/
#include "src/stdlib.cl";
struct _test{
    Pitch test;
};
int main(){
   _test a;
   a.test = 1^4;
   print(a.test);
}
```

```
1^4
```

Test: struct with pitch array

```
/*Kunjian Liao*/
#include "src/stdlib.cl";
struct _test{
```

```
    Pitch[] test;
};
int main(){
   _test a;
   a.test = [1^4, 1#4];
   print(a.test[0]);
}
```

```
1^4
```

Test: Struct with multiple arrays

```
/*Kunjian Liao*/
#include "src/stdlib.cl";
struct _test{
    Pitch[] test;
    int[] test2;
};
int main(){
   _test a;
   a.test = [1^4, 1#4];
   a.test2 = [1, 2, 3];
   print(a.test[0]);
   print(a.test2[0]);
}
```

```
1^4
1
```

Test: Include struct from external file

```
/*Kunjian Liao*/
#include "src/stdlib.cl";
struct _test{
    Pitch test;
```

```
    int test1;
};
```

```
/*Kunjian Liao*/
#include "test_kj/include_struct_helper.cl";
int main(){
   _test a;
   a.test = 1^4;
   print(a.test);
}
```

```
1^4
```

Test: Include function from external file

```
/*Kunjian Liao*/
#include "src/stdlib.cl";
int add(int a, int b){
   int x = a+b;
   return x;
}
```

```
/*Kunjian Liao*/
#include "test_kj/include_function_helper.cl";
int main(){
   int a = 1;
   int b = 2;
   int x = add(a,b);
   print(x);
   return 0;
}
```

```
3
```

Test: Array operation with different object

```
int main(){
   int[4] int_array = [1,2,3,4];
   bool[4] bool_array = [true,false,true,false];
   String[4] str_array = ["i","like","coding","ocaml"];
   Pitch[4] pitch_array = [1^4,2^4,3#4,4#4];
   float[4] float_array = [1.1,2.3,3.4,5.6];
   return 0;
}
```

```
int main(){
   int i;
   int[4] int_array = [1,2,3,4];
   bool[4] bool_array = [true,false,true,false];
   String[4] str_array = ["i","like","coding","ocaml"];
   Pitch[4] pitch_array = [1^4,2^4,3#4,4#4];
   float[4] float_array = [1.1,2.3,3.4,5.6];
   int_array[2]=30;
   bool_array[1]=false;
   str_array[3]="java";
   pitch_array[1]=1^4;
   float_array[3]=2.2;
   for (i=0;i<4;i=i+1){
      print(int_array[i]);
      print(str_array[i]);
      print(pitch_array[i]);
      print(float_array[i]);
   }
   return 0;
}
```

```
1
i
1^4
1.1
2
```

```
like
1^4
2.3
30
coding
3#4
3.4
4
java
4#4
2.2
```

Test: Array assignment

```
int main(){
   int i;
   int[4] int_array = [1,2,3,4];
   int[4] dup = int_array;
   dup[2] = 20;
   for (i=0;i<size(dup);i=i+1){
      print(dup[i]);
   }
   for (i=0;i<size(int_array);i=i+1){
      print(int_array[i]);
   }

   return 0;
}
```

```
1
2
20
4
1
2
20
```

```
4
```

```
int main(){
   int i;
   int[4] int_array = [1,2,3,4];
   int[5] dup = int_array@[1];
   dup[2] = 20;
   for (i=0;i<size(dup);i=i+1){
      print(dup[i]);
   }
   for (i=0;i<size(int_array);i=i+1){
      print(int_array[i]);
   }

   return 0;
}
```

```
1
2
20
4
1
1
2
3
4
```

6.2 Integration Test

Test: Bubble Sort

```
/*Author: Yipeng Zhang*/
int[] bubble_sort(int[] Arr, int sz){
   int i;
   int j;
   int temp;
```

```
   for (i=0;i<sz;i=i+1){
      for (j=i+1;j<sz;j=j+1){
         if (Arr[i]>Arr[j]){
            temp = Arr[i];
            Arr[i] = Arr[j];
            Arr[j] = temp;
         }
      }
   }
   return Arr;
}


int main()
{
    int i;
    int[7] sorted;
   int[7] Arr = [random(50),random(50),random(50),random(50),
                 random(50),random(50),random(50)];
    print("original array");
    for (i=0;i<size(Arr);i=i+1){
        print(Arr[i]);
    }
   sorted = bubble_sort(Arr, 7);
    print("sorted array");
   for (i=0;i<size(sorted);i=i+1){
      print(sorted[i]);
   }
   return 0;
}
```

```
original array
7
49
23
8
30
```

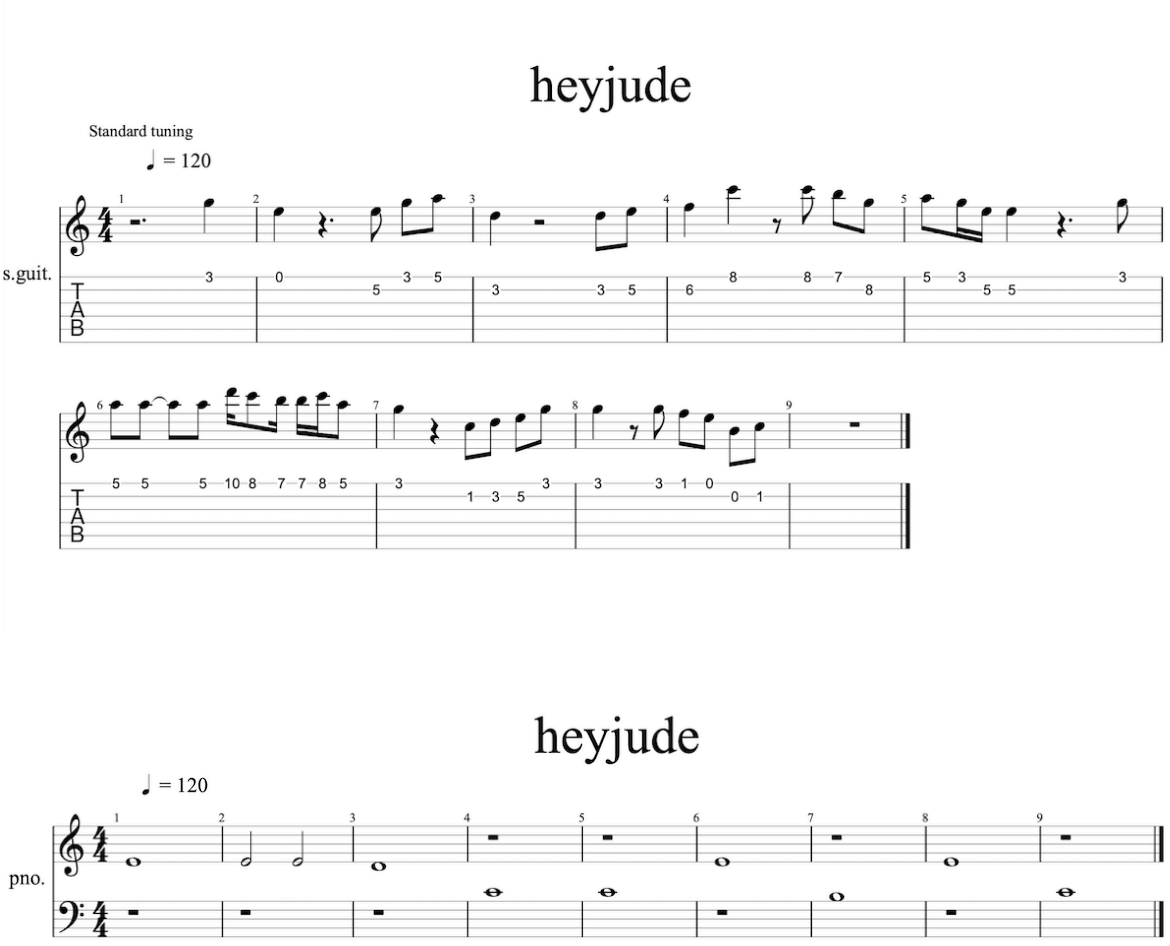
```
22
44
sorted array
7
8
22
23
30
44
49
```

Test: Multiple tracks, Hey Jude

```
/*Author: Yuanji Huang*/
#include "src/stdlib.cl";
int main()
{
   _track track1;
    _track track2;

    Pitch[49] melody =
    [1b1,1b1,5^4,
     3^4,1b1,1b1,3^4,5^4,6^4,
     2^4,1b1,1b1,2^4,3^4,
     4^4,1^5,1b1,1^5,7^4,5^4,
     6^4,5^4,3^4,3^4,1b1,1b1,5^4,
     6^4,6^4,6^4,2^5,1^5,7^4,7^4,1^5,6^4,
     5^4,1b1,1^4,2^4,3^4,5^4,
     5^4,1b1,5^4,4^4,3^4,7^3,1^4];

    int[49] rhythm =
    [2,4,4,
     4,4,8,8,8,8,
     4,4,4,8,8,
     4,4,8,8,8,8,
     8,16,16,4,4,8,8,
     8,4,8,16,8,16,16,16,8,
```

```
     4,4,8,8,8,8,
     4,8,8,8,8,8,8];

    Pitch[10] chorus = [3^4,3^4,3^4,2^4,1^4,1^4,3^4,7^3,3^4,1^4];
    int[10] chourus_rhythm = [1,2,2,1,1,1,1,1,1,1];
   track1 = build_track(25, 49, melody, rhythm);
    track2 = build_track(1, 10, chorus, chourus_rhythm);
   generate_music(track1);
    generate_music(track2);
   return 0;
}
```

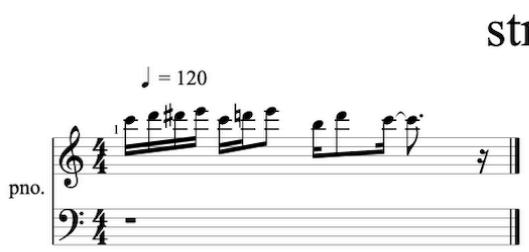
Music sheet of generated MIDI file:



Test: Record daily music inspiration by using struct

```
/*Author: Kunjian Liao*/
#include "src/stdlib.cl";
struct _inspiration{
    String name;
    Pitch[] melody;
    int[] rhythm;
    int rating;
};
int main(){
   _inspiration May11th;
    _inspiration May12th;
    _track my_choice;
   May11th.melody = [3^5, 2#5, 3^5, 2#5, 3^5, 7^4, 2^5, 1^5, 6^4];
   May11th.rhythm = [16, 16, 16, 16, 16, 16, 16, 16, 8];
    May11th.rating = 1;
   May12th.melody = [1^6, 2^6, 2#6, 3^6, 1^6, 2^6, 3^6, 7^5, 2^6, 1^6];
   May12th.rhythm = [16, 16, 16, 16, 16, 16, 8, 16, 8, 4];
   May12th.rating = 2;
    May11th.name = "For Alice";
    May12th.name = "Entertainer";

   if(May12th.rating < May11th.rating){
        my_choice = build_track(1,9,May11th.melody,May11th.rhythm);
   }
   else{
      my_choice = build_track(1,10,May12th.melody,May12th.rhythm);
   }
   generate_music(my_choice);
}
```

Music sheet of generated MIDI file:

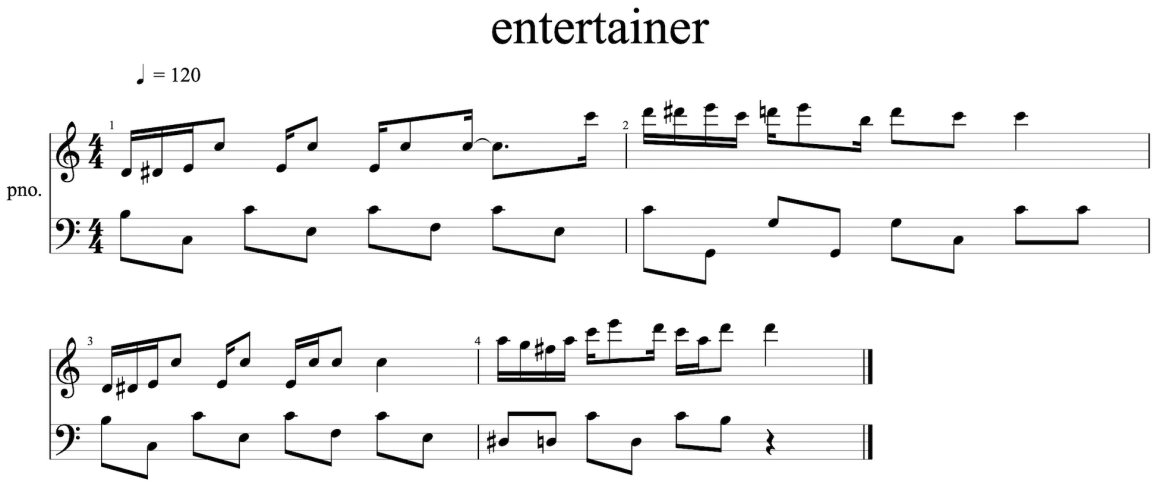


Test: Track in parallel, The Entertainer, with chorus. MIDI available on Github

```
/*Kunjian Liao*/
#include "src/stdlib.cl";
int main(){
   _track entertainer;
   _track chorus;

   Pitch[9] melody1 = [2^4, 2#4, 3^4, 1^5, 3^4, 1^5, 3^4, 1^5, 1^5];
   int[9] rhythm1 = [16, 16, 16, 8, 16, 8, 16, 8, 4];
   Pitch[11] melody2 = [1^6, 2^6, 2#6, 3^6, 1^6, 2^6, 3^6, 7^5, 2^6, 1^6, 1^6];
   int[11] rhythm2 = [16, 16, 16, 16, 16, 16, 8, 16, 8, 8, 4];
   Pitch[11] melody3 = [6^5, 5^5, 4#5, 6^5, 1^6, 3^6, 2^6, 1^6, 6^5, 2^6, 2^6];
   int[11] rhythm3 = [16, 16, 16, 16, 16, 8, 16, 16, 16, 8, 4];
   Pitch[10] melody4 = [2^4, 2#4, 3^4, 1^5, 3^4, 1^5, 3^4, 1^5, 1^5, 1^5];
   int[10] rhythm4 = [16, 16, 16, 8, 16, 8, 16, 16, 8, 4];
   Pitch[41] melody = melody1@melody2@melody4@melody3;
   int[41] rhythm = rhythm1@rhythm2@rhythm4@rhythm3;

   Pitch[16] c_melody1 = [7^3, 1^3, 1^4, 3^3, 1^4, 4^3, 1^4, 3^3, 1^4, 5^2,
      5^3, 5^2, 5^3, 1^3, 1^4, 1^4];
   int[16] c_rhythm1 = [8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8];
   Pitch[14] c_melody2 = [7^3, 1^3, 1^4, 3^3, 1^4, 4^3, 1^4, 3^3, 3b3, 2^3,
      1^4, 2^3, 1^4, 7^3];
   int[14] c_rhythm2 = [8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8];
   Pitch[30] c_melody = c_melody1@c_melody2;
   int[30] c_rhythm = c_rhythm1@c_rhythm2;
```

```
    entertainer = build_track(1, 41, melody, rhythm);
    chorus = build_track(1, 30, c_melody, c_rhythm);

    generate_music(entertainer);
    generate_music(chorus);
    return 0;
}
```

Music sheet of generated MIDI file:



7 LESSONS LEARNED

Yipeng Zhang: I think the most important lesson for me is how hard it is to design a complex programming language like C++ or python. Every tiny bit of convenience for the programmer could take dozens of hours of works on the compilers side, as little as an array access function. Huge respect for them who contributed behind the curtain.
Another lesson for me is the importance of team work and communication. I have twisted my heads around how to accommodate the variable declaration and assignments which is take as granted by most modern programming language. It's very hard to change the whole structure from ast to codegen and I have always wanted to take some shortcut. Until Yuanji and Kunjian helped me with this and we fixed this problem by updating as little as possible,

the bind declaration in ast. We fixed this issue in a few hours which has entangled me for more than a week. Team work is really useful in such a project and I love my team!

Yuanji Huang: Never takes the convenience features of modern programming language as granted! When I wrote down some fancy features in proposal, I have never thought that I will spend several hours on writing a simple get size function... Type inference, indent block, prompt and other common features in Python could be very hard to implement. Another lesson I learn is never try to move things that should be done in later stage to semantic checking, otherwise you will counter with causal issues. For example, we used to try updating array size in symbol table during sematic checking, but unfortunately array size would change during program execution. I also have a deep understanding in LLVM. Before this class, I have no idea what is LLVM but with this project I understand the mechanism of how LLVM build IR code and how it transform to executable file.

Kunjian Liao: One significant change happened on me after finishing this project is that I began to cherish all the features and functionalities that current high-level programming languages offer. When I transited from C++ to Python a few years ago, I was confused about why the inventor of C++ made many troublesome rules, like I have to specify the size of an array when declare it, the use of pointers, etc. Now, I begin to understand how difficult to implement features seemingly intuitive, such as type inference, flexible array. In this project, even keeping track of the size of arrays took us tons of time. Also, I now understand how the programming languages are understood by computer, which may help me improve my programming skills.
Another award that I did not expect is that I picked up my piano skills again. I stopped playing piano nearly one decade ago. During this period working on this music composing programming language project, I learnt much music theory, and listened dozens of classical music written by Chopin, Bach, Mozart, etc. This experience triggered my passion on piano, and now I practice the piano an hour a day.

8 APPENDIX

8.1 ast.ml

```
(* Abstract Syntax Tree and functions for printing it *)
(* Author: Yipeng Zhang, Yuanji Huang, Kunjian Liao*)
```

```
type op = Add | Sub | Mult | Div | Con | Equal | Neq | Less | Leq | Greater |
    Geq |
          And | Or | Mod

type uop = Neg | Not

type typ = Int | Bool | Float | Void | String | Pitch | Array of typ * int |
    Struct of string

type expr =
    Literal of int
  | Fliteral of string
  | BoolLit of bool
  | Sliteral of string
  | Pliteral of string
  | ArrayLit of expr list
  | ArrayAccess of expr * expr
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of expr * expr
  | Call of string * expr list
  | StructAccess of expr * string
  | Noexpr

type bind_value = typ * string * expr

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

type func_decl = {
    typ : typ;
```

```
    fname : string;
    formals : bind_value list;
    locals : bind_value list;
    body : stmt list;
  }

type struct_decl = {
    members: bind_value list;
    struct_name: string;
  }

type include_decl = Include of string

type program = {
    globals: bind_value list;
    functions: func_decl list;
    structs: struct_decl list;
    includes: include_decl list;
}




(* Pretty-printing functions *)
(* todo: support struct print*)
let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Con -> "@"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
```

```
  | And -> "&&"
  | Or -> "||"
  | Mod -> "%"

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

let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | Fliteral(l) -> l
  | Sliteral(l) -> l
  | Pliteral(l) -> l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | ArrayLit(el) -> "[" ^ String.concat ", " (List.map string_of_expr el) ^ "]"
  | ArrayAccess(s,e) -> "Array_access" ^ " "^ string_of_expr s ^ " "
     ^string_of_expr e
  | 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(e1, e2) -> string_of_expr e1 ^ " = " ^ string_of_expr e2
  | Call(f, el) ->
     f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | StructAccess(s, n) ->
    (string_of_expr s) ^ "." ^ n
  | Noexpr -> ""

let rec string_of_typ = function
    Int -> "int"
  | Bool -> "bool"
  | Float -> "float"
  | Void -> "void"
  | String -> "string"
  | Array (x,size) -> (string_of_typ x) ^ "["^(string_of_int size)^"]"
  | Pitch -> "Pitch"
```

```
  | Struct(id) -> id

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_vdecl (t, id, _) = string_of_typ t ^ " " ^ id ^ ";\n"

let get_second (_, id, _) = id

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

let string_of_sdecl sdecl =
  "struct " ^ sdecl.struct_name ^ "\n" ^
  "{\n" ^
  String.concat "" (List.map string_of_vdecl sdecl.members) ^
  "}\n"

let string_of_include = function
  Include(s) -> "#include<" ^ s ^ ">;\n"
```

```
let string_of_program program =
  String.concat "" (List.map string_of_include program.includes) ^
  String.concat "\n" (List.map string_of_vdecl program.globals) ^
  String.concat "\n" (List.map string_of_fdecl program.functions) ^
  String.concat "\n" (List.map string_of_sdecl program.structs)
```

8.2 scanner.mll

```
(* Ocamllex scanner for CAML *)
(* Author: Yipeng Zhang, Yuanji Huang, Kunjian Liao*)
{ open Parser }

let digit = ['0' - '9']
let digits = digit+

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"    { comment lexbuf }           (* Comments *)
| '('     { LPAREN }
| ')'     { RPAREN }
| '{'     { LBRACE }
| '}'     { RBRACE }
| '['     { LSQUARE }
| ']'     { RSQUARE }
| ';'     { SEMI }
| ','     { COMMA }
| '+'     { PLUS }
| '-'     { MINUS }
| '*'     { TIMES }
| '/'     { DIVIDE }
| '@'     { CONCAT }
| '%'   { MOD }
| '='     { ASSIGN }
| ':'     { COLON }
| '.'     { DOT }
| '''      { APOSTROPHE }
```

```
| '"'      { QUOTE }
| '"' (('\\' '"'| [^'"'])* as str) '"' { SLIT(Scanf.unescaped str) }
| "=="    { EQ }
| "!="    { NEQ }
| '<'     { LT }
| "<="    { LEQ }
| ">"     { GT }
| ">="    { GEQ }
| "&&"    { AND }
| "||"    { OR }
| "!"     { NOT }
| "if"    { IF }
| "else"  { ELSE }
| "for"   { FOR }
| "while" { WHILE }
| "return" { RETURN }
| "int"   { INT }
| "bool"  { BOOL }
| "float" { FLOAT }
| "String" { STR }
| "Pitch" { PITCH }
| "struct" { STRUCT }
| "void"  { VOID }
| "true"  { BLIT(true) }
| "false" { BLIT(false) }
| "#"   { POUND }
| "include"   { INCLUDE }
| "Array"    { ARRAY }
| digits as lxm { LITERAL(int_of_string lxm) }
| digits '.' digit* ( ['e' 'E'] ['+' '-']? digits )? as lxm { FLIT(lxm) }
| ['1'-'7'] ['b' '#' '^'] ['0'-'8']              as lxm { PLIT(lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| ['_']['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { STLIT(lxm)}
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

and comment = parse
```

```
  "*/" { token lexbuf }
| _    { comment lexbuf }
```

8.3 Parser.mly

```
/* Ocamlyacc parser for CAML */
(* Author: Yipeng Zhang, Yuanji Huang, Kunjian Liao*)
%{
open Ast
%}

%token QUOTE APOSTROPHE COLON LSQUARE RSQUARE INCLUDE
%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA PLUS MINUS TIMES DIVIDE CONCAT
    MOD ASSIGN POUND
%token NOT EQ NEQ LT LEQ GT GEQ AND OR DOT
%token RETURN IF ELSE FOR WHILE INT BOOL FLOAT VOID STR PITCH STRUCT
%token <int> LITERAL
%token <bool> BLIT
%token <string> ID FLIT SLIT PLIT STLIT
%token ARRAY
%token EOF

%start program
%type <Ast.program> program

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE MOD
%right CONCAT
%right NOT
```

```
%left DOT
%left LSQUARE RSQUARE
%%

program:
   decls EOF { $1 }

decls:
   /* nothing */ { {includes=[]; globals=[]; functions=[]; structs=[]} }
 | decls idecl { {includes = ($2 :: $1.includes); globals = $1.globals;
     functions = $1.functions; structs = $1.structs} }
 | decls vdecl { {includes = $1.includes; globals = ($2 :: $1.globals);
     functions = $1.functions; structs = $1.structs} }
 | decls fdecl { {includes = $1.includes; globals = $1.globals; functions = ($2
     :: $1.functions); structs = $1.structs} }
 | decls sdecl { {includes = $1.includes; globals = $1.globals; functions =
     $1.functions; structs = ($2 :: $1.structs)} }

idecl:
  POUND INCLUDE SLIT SEMI { Include($3) }

vdecl:
   typ ID SEMI { ($1, $2, Noexpr) }
   | typ ID ASSIGN expr SEMI { ($1, $2, $4) }

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

fdecl:
    typ ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE
      { { typ = $1;
          fname = $2;
          formals = List.rev $4;
          locals = List.rev $7;
          body = List.rev $8 } }
```

```
sdecl:
  STRUCT STLIT LBRACE vdecl_list RBRACE SEMI
    {
    {
      struct_name = $2;
      members = $4;
    }}

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

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

typ:
    INT   { Int  }
  | BOOL { Bool }
  | FLOAT { Float }
  | VOID { Void }
  | STR  { String }
  | PITCH { Pitch }
  | STLIT { Struct($1) }
  | typ LSQUARE LITERAL RSQUARE { Array($1,$3) }
  | typ LSQUARE RSQUARE { Array($1,10) }

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

stmt:
   expr SEMI                              { Expr $1            }
   | RETURN expr_opt 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:
     LITERAL        { Literal($1)          }
   | FLIT            { Fliteral($1)         }
   | BLIT           { BoolLit($1)          }
   | SLIT           { Sliteral($1)         }
   | PLIT           { Pliteral($1)         }
   | 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 CONCAT expr { Binop($1, Con, $3) }
   | expr MOD   expr { Binop($1, Mod, $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 DOT ID    { StructAccess($1, $3) }
   | expr LSQUARE expr RSQUARE { ArrayAccess($1, $3) }
   | MINUS expr %prec NOT { Unop(Neg, $2)  }
   | NOT expr       { Unop(Not, $2)        }
   | expr ASSIGN expr { Assign($1, $3)     }
   | ID LPAREN args_opt RPAREN { Call($1, $3) }
   | LPAREN expr RPAREN { $2               }
```

```
   | LSQUARE args_opt RSQUARE { ArrayLit($2) }

args_opt:
   /* nothing */ { [] }
   | args_list { List.rev $1 }

args_list:
   expr                  { [$1] }
   | args_list COMMA expr { $3 :: $1 }
```

8.4 sast.mly

```
(* Semantically-checked Abstract Syntax Tree and functions for printing it *)
(* Author: Yipeng Zhang, Yuanji Huang, Kunjian Liao*)
open Ast

type sexpr = typ * sx
and sx =
    SLiteral of int
  | SFliteral of string
  | SBoolLit of bool
  | SSliteral of string
  | SPliteral of string
  | SId of string
  | SBinop of sexpr * op * sexpr
  | SUnop of uop * sexpr
  | SAssign of sexpr * sexpr
  | SCall of string * sexpr list
  | SStructAccess of sexpr * string
  | SArrayLit of sexpr list
  | SArrayAccess of sexpr * sexpr
  | SNoexpr

type sstmt =
    SBlock of sstmt list
  | SExpr of sexpr
```

```
  | SReturn of sexpr
  | SIf of sexpr * sstmt * sstmt
  | SFor of sexpr * sexpr * sexpr * sstmt
  | SWhile of sexpr * sstmt

type sfunc_decl = {
    styp : typ;
    sfname : string;
    sformals : bind_value list;
    slocals : bind_value list;
    sbody : sstmt list;
  }

type sstruct_decl = {
    smembers: bind_value list;
    sstruct_name: string;
  }

type sprogram = {
    sglobals: bind_value list;
    sfunctions: sfunc_decl list;
    sstructs: sstruct_decl list;
}

(* Pretty-printing functions *)

let rec string_of_sexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
    SLiteral(l) -> string_of_int l
  | SSliteral(l) -> l
  | SBoolLit(true) -> "true"
  | SBoolLit(false) -> "false"
  | SFliteral(l) -> l
  | SPliteral(l) -> l
  | SId(s) -> s
  | SBinop(e1, o, e2) ->
      string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
```

```
  | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
  | SAssign(e1, e2) -> string_of_sexpr e1 ^ " = " ^ string_of_sexpr e2
  | SCall(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
  | SStructAccess(s, n) ->
    (string_of_sexpr s) ^ "." ^ n
  | SArrayLit(el) -> "[" ^ String.concat ", " (List.map string_of_sexpr el) ^
      "]"
  | SArrayAccess(e1, e2) -> "sarray_access "^string_of_sexpr e1 ^ " "
      ^string_of_sexpr e2
  | SNoexpr -> ""
            ) ^ ")"

let rec string_of_sstmt = function
    SBlock(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
  | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
  | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n";
  | SIf(e, s, SBlock([])) ->
      "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
  | SIf(e, s1, s2) -> "if (" ^ string_of_sexpr e ^ ")\n" ^
      string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
  | SFor(e1, e2, e3, s) ->
      "for (" ^ string_of_sexpr e1 ^ " ; " ^ string_of_sexpr e2 ^ " ; " ^
      string_of_sexpr e3 ^ ") " ^ string_of_sstmt s
  | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt s

let get_second (_, id, _) = id

let string_of_sfdecl fdecl =
  string_of_typ fdecl.styp ^ " " ^
  fdecl.sfname ^ "(" ^ String.concat ", " (List.map get_second fdecl.sformals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.slocals) ^
  String.concat "" (List.map string_of_sstmt fdecl.sbody) ^
  "}\n"
```

```
let string_of_ssdecl sdecl =
  "struct " ^ sdecl.sstruct_name ^ "\n" ^
  "{\n" ^
  String.concat "" (List.map string_of_vdecl sdecl.smembers) ^
  "}\n"

let string_of_sprogram sprogram =
  String.concat "" (List.map string_of_vdecl sprogram.sglobals) ^ "\n" ^
  String.concat "\n" (List.map string_of_sfdecl sprogram.sfunctions) ^
  String.concat "\n" (List.map string_of_ssdecl sprogram.sstructs)
```

8.5 semant

```
(* Semantic checking for the CAML compiler *)
(* Author: Yipeng Zhang, Yuanji Huang, Kunjian Liao*)

open Ast
open Sast

module StringMap = Map.Make(String)

(* Semantic checking of the AST. Returns an SAST if successful,
   throws an exception if something is wrong.

   Check each global variable, then check each function *)

let check program =

  (* Verify a list of bindings has no void types or duplicate names *)
  let check_binds (kind : string) (binds : bind_value list) =
    List.iter (function
       (Void, b, _) -> raise (Failure ("illegal void " ^ kind ^ " " ^ b))
      | (_,_,_) -> ())
    binds;

    let rec dups = function
```

```
      [] -> ()
    |  ((_,n1,_) :: (_,n2,_) :: _) when n1 = n2 ->
   raise (Failure ("duplicate " ^ kind ^ " " ^ n1))
    | _ :: t -> dups t
  in dups (List.sort (fun (_,a,_) (_,b,_) -> compare a b) binds)
in

(**** Check global variables ****)
check_binds "global" program.globals;

(**** Check functions ****)

let rec add_all_include (all_program: Ast.program) : Ast.program =
  let add_current_include (current_program: Ast.program)(Ast.Include(incl) :
      Ast.include_decl) : Ast.program=
    let file_in = open_in incl in
    let lexbuf = Lexing.from_channel file_in in
    let import_program = Parser.program Scanner.token lexbuf in
    let (after_import_program: Ast.program) = add_all_include import_program
        in
    ignore(close_in file_in);
    {
      globals = current_program.globals;
      functions = after_import_program.functions@current_program.functions;
      structs = after_import_program.structs@current_program.structs;
      includes = after_import_program.includes
    }
  in List.fold_left add_current_include all_program all_program.includes
in

let program_with_include = add_all_include program
in

(* Collect function declarations for built-in functions: no bodies *)
let built_in_decls =
  let add_bind map (ftype, name, param_list) = StringMap.add name {
    typ = ftype;
```

```
    fname = name;
    formals = param_list;
    locals = [];
    body = []
  } map in
  let builtin_funcs =
  [(Void, "print", [(Int,"x",Noexpr)]);
    (Void, "printb", [(Bool,"x",Noexpr)]);
    (Void, "printbig", [(Int,"x",Noexpr)]);
    (Void, "prints", [(String,"x",Noexpr)]);
    (Void, "printp", [(Pitch,"x",Noexpr)]);
    (Int, "pitch_to_int", [(Pitch,"x",Noexpr)]);
    (Int, "random", [(Int,"x",Noexpr)]);
    (Pitch, "up_8", [(Pitch,"x",Noexpr)]);
    (Pitch, "down_8", [(Pitch,"x",Noexpr)])]
  in
  List.fold_left add_bind StringMap.empty builtin_funcs
in
(* Add function name to symbol table *)
let add_func map fd =
  let built_in_err = "function " ^ fd.fname ^ " may not be defined"
  and dup_err = ":duplicate function " ^ fd.fname
  and make_err er = raise (Failure er)
  and n = fd.fname (* Name of the function *)
  in match fd with (* No duplicate functions or redefinitions of built-ins *)
       _ when StringMap.mem n built_in_decls -> make_err built_in_err
     | _ when StringMap.mem n map -> make_err dup_err
     | _ -> StringMap.add n fd map
in

(* Collect all function names into one symbol table *)
let function_decls = List.fold_left add_func built_in_decls
    program_with_include.functions
in

(* Return a function from our symbol table *)
let find_func s =
```

```
  try StringMap.find s function_decls
  with Not_found -> raise (Failure ("unrecognized function " ^ s))
in

let add_struct map sd =
  let n = sd.struct_name
  and dup_err = "duplicate struct " ^ sd.struct_name
  and make_err er = raise (Failure er)
  in match sd with
      _ when StringMap.mem n map -> make_err dup_err
    | _ -> StringMap.add n sd map
in
(* Collect all struct names into one symbol table *)
let struct_decls = List.fold_left add_struct StringMap.empty
    program_with_include.structs
in

let find_struct s = try StringMap.find s struct_decls
    with Not_found -> raise (Failure ("unrecognized struct " ^ s))
in

let _ = find_func "main" in (* Ensure "main" is defined *)

let check_function func =
  (* Make sure no formals or locals are void or duplicates *)
  check_binds "formal" func.formals;
  check_binds "local" func.locals;

  let add_assign (stmts:stmt list) (local:bind_value) = match local with
    (_,_,Noexpr) -> stmts
   |(_,n,e) -> Expr(Assign((Id(n)),e))::stmts
  in
  let new_body = List.fold_left add_assign func.body (List.rev func.locals)
  in

  (* Raise an exception if the given rvalue type cannot be assigned to
     the given lvalue type *)
```

```
let check_argument lvaluet rvaluet err = match (lvaluet,rvaluet) with
  (Array(t1 ,_), Array(t2, _)) -> if t1 = t2 then rvaluet else raise
      (Failure err)
  | _ -> if lvaluet = rvaluet then rvaluet else raise (Failure err)
in
let check_equal lvaluet rvaluet = match (lvaluet,rvaluet) with
  (Array(t1,_),Array(t2,_)) -> t1 = t2
  | _ -> lvaluet = rvaluet
in
(* Build local symbol table of variables for this function *)
let symbols = List.fold_left (fun m (ty, name, _) -> StringMap.add name ty
    m)
            StringMap.empty (program_with_include.globals @ func.formals
                @ func.locals )
in

(* Return a variable from our local symbol table *)
let type_of_identifier s =
  try StringMap.find s symbols
  with Not_found -> raise (Failure ("undeclared identifier " ^ s))
in

(* Return a semantically-checked expression, i.e., with a type *)
let rec expr = function
    Literal l -> (Int, SLiteral l)
  | Fliteral l -> (Float, SFliteral l)
  | BoolLit l -> (Bool, SBoolLit l)
  | Sliteral l -> (String, SSliteral l)
  | Pliteral l -> (Pitch, SPliteral l)
  | Noexpr    -> (Void, SNoexpr)
  | Id s      -> (type_of_identifier s, SId s)


  | Assign(left_e, right_e) as ex ->
      let (lt, e1) = expr left_e
      and (rt, e2) = expr right_e in
      let err = "illegal assignment " ^ string_of_typ lt ^ " = " ^
```

```
      string_of_typ rt ^ " in " ^ string_of_expr ex
    in (check_argument lt rt err, SAssign((lt, e1), (rt, e2)))

| Unop(op, e) as ex ->
    let (t, e') = expr e in
    let ty = match op with
      Neg when t = Int || t = Float -> t
    | Not when t = Bool -> Bool
    | _ -> raise (Failure ("illegal unary operator " ^
                           string_of_uop op ^ string_of_typ t ^
                           " in " ^ string_of_expr ex))
    in (ty, SUnop(op, (t, e')))



| Binop(e1, op, e2) as e ->
  let (t1, e1') = expr e1
  and (t2, e2') = expr e2 in
  let same = check_equal t1 t2 in
    let ty = (match op with
      Add | Sub | Mult | Div | Mod when same && t1 = Int -> Int
    | Add | Sub | Mult | Div | Mod when same && t1 = Float -> Float
    | Equal | Neq          when same              -> Bool
    | Less | Leq | Greater | Geq
             when same && (t1 = Int || t1 = Float) -> Bool
    | And | Or when same && t1 = Bool -> Bool
    | Con when same -> (match (t1,t2) with (Array(typ, sz1), Array(_,
       sz2))-> Array(typ,sz1+sz2)
                                       | _-> raise (Failure"illegal binary
                                          operator concat"))
    | _ -> raise (
      Failure ("illegal binary operator " ^
                string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                string_of_typ t2 ^ " in " ^ string_of_expr e)))
    in (ty, SBinop((t1, e1'), op, (t2, e2')))
    (* Determine expression type based on operator and operand types *)
| Call(fname, args) as call ->
```

```
    (match fname with
    | "size" | "print" ->
       let args' = List.map expr args in
              (Int, SCall (fname, args'))
    | _ ->
      let fd = find_func fname in
      let param_length = List.length fd.formals in
      if List.length args != param_length then
        raise (Failure ("expecting " ^ string_of_int param_length ^
                        " arguments in " ^ string_of_expr call))
      else let check_call (ft, _, _) e =
        let (et, e') = expr e in
        let err = "illegal argument found " ^ string_of_typ et ^
          " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e
        in (check_argument ft et err, e')
      in
      let args' = List.map2 check_call fd.formals args
    in (fd.typ, SCall(fname, args')))
| StructAccess(s, m) as sacc ->
  let ss = expr s in
  let s_type = fst ss in
  let sd = find_struct (string_of_typ s_type)
  in
    let members = List.fold_left (fun m (t,n,_) -> StringMap.add n t m)
       StringMap.empty sd.members in
    (* Iterate through the members of the struct; if name found, return
       its type, else fail *)
    (try let tem = StringMap.find m members in
     (tem, SStructAccess(ss,m))
     with Not_found -> raise (Failure ("illegal member " ^ m ^ " of struct
        " ^ string_of_expr sacc)))
| ArrayLit vals ->
   let length = List.length vals in
   let (t',_) = expr (List.hd vals) in
   let map_func lit = expr lit in
   let vals' = List.map map_func vals in
   (* TODO: check that all vals are of the same type *)
```

```
      (Array (t',length), SArrayLit(vals'))
  | ArrayAccess (var, idx) ->
    let (t1, se1) = expr var in
    let (t2, se2) = expr idx in

    let t3 = match t1 with
        Array(t, _) -> t
      | _ -> raise (Failure ("not an array"))
    in
    if t2 = Int then (t3, SArrayAccess((t1, se1), (t2, se2)))
    else raise (Failure ("can't access array with non-integer type"))
in

let check_bool_expr e =
  let (t', e') = expr e
  and err = "expected Boolean expression in " ^ string_of_expr e
  in if t' != Bool then raise (Failure err) else (t', e')
in

(* Return a semantically-checked statement i.e. containing sexprs *)
let rec check_stmt = function
    Expr e -> SExpr (expr e)
  | If(p, b1, b2) -> SIf(check_bool_expr p, check_stmt b1, check_stmt b2)
  | For(e1, e2, e3, st) ->
     SFor(expr e1, check_bool_expr e2, expr e3, check_stmt st)
  | While(p, s) -> SWhile(check_bool_expr p, check_stmt s)
  | Return e -> let (t, e') = expr e in
    if t = func.typ then SReturn (t, e')
    else raise (Failure ("return gives " ^ string_of_typ t ^ " expected " ^
     string_of_typ func.typ ^ " in " ^ string_of_expr e))

   (* A block is correct if each statement is correct and nothing
      follows any Return statement. Nested blocks are flattened. *)
  | Block sl ->
      let rec check_stmt_list = function
          [Return _ as s] -> [check_stmt s]
        | Return _ :: _ -> raise (Failure "nothing may follow a return")
```

```
          | Block sl :: ss -> check_stmt_list (sl @ ss) (* Flatten blocks *)
          | s :: ss       -> check_stmt s :: check_stmt_list ss
          | []            -> []
        in SBlock(check_stmt_list sl)

  in (* body of check_function *)
  { styp = func.typ;
    sfname = func.fname;
    sformals = func.formals;
    slocals = func.locals;
    sbody = match check_stmt (Block new_body) with
 SBlock(sl) -> sl
    | _ -> raise (Failure ("internal error: block didn't become a block?"))
  }
in

let check_struct struc =
{
  sstruct_name = struc.struct_name;
  smembers = struc.members;
}
in

{
  sglobals = program_with_include.globals;
  sfunctions = List.map check_function program_with_include.functions;
  sstructs = List.map check_struct program_with_include.structs;
}
```

8.6 codegen.ml

```
(* Code generation: translate takes a semantically checked AST and
produces LLVM IR

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

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

Detailed documentation on the OCaml LLVM library:

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

*)
(* Author: Yipeng Zhang, Yuanji Huang, Kunjian Liao*)

module L = Llvm
module A = Ast
open Sast

module StringMap = Map.Make(String)

(* translate : Sast.program -> Llvm.module *)


let translate program =
  let globals = program.sglobals
  and functions = program.sfunctions
  and structs = program.sstructs
  in
  let context   = L.global_context () in

  (* Create the LLVM compilation module into which
     we will generate code *)
  let the_module = L.create_module context "CAML" in

  (* Get types from the context *)
  let i32_t     = L.i32_type   context
  and i8_t      = L.i8_type    context
  and i1_t      = L.i1_type    context
  and float_t   = L.double_type context
  and void_t    = L.void_type  context in
  let struct_t n = L.named_struct_type context n in
```

```
let str_t     = L.pointer_type i8_t
and pitch_t   = L.pointer_type i8_t
in
(* Return the LLVM type for a CAML type *)
let rec ltype_of_primitive = function
    A.Int   -> i32_t
  | A.Bool  -> i1_t
  | A.Float -> float_t
  | A.String -> str_t
  | A.Void  -> void_t
  | A.Pitch -> pitch_t
  | A.Struct n -> struct_t n
  | A.Array (array_typ,_) -> L.pointer_type (ltype_of_primitive array_typ)
in
(* Create a map of global variables after creating each *)
let global_vars : L.llvalue StringMap.t =
  let global_var m (t, n, _) =
    let init = match t with
        A.Float -> L.const_float (ltype_of_primitive t) 0.0
      | _ -> L.const_int (ltype_of_primitive t) 0
    in StringMap.add n (L.define_global n init the_module) m in
  List.fold_left global_var StringMap.empty globals
in
let str_to_int : L.lltype =
    L.var_arg_function_type i32_t [| str_t |]
and int_to_int : L.lltype =
    L.var_arg_function_type i32_t [| i32_t |]
and pitch_to_int : L.lltype =
    L.var_arg_function_type i32_t [|pitch_t|]
and pitch_to_pitch : L.lltype =
    L.var_arg_function_type i32_t [|pitch_t|] in

let printf_func : L.llvalue =
    L.declare_function "printf" str_to_int the_module
and prints_func : L.llvalue =
    L.declare_function "puts" str_to_int the_module
and pitch_to_int_func : L.llvalue =
```

```
    L.declare_function "pitch_to_int" pitch_to_int the_module
and random_func : L.llvalue =
    L.declare_function "ran" int_to_int the_module
and up8_func : L.llvalue =
    L.declare_function "up_8" pitch_to_pitch the_module
and down8_func : L.llvalue =
    L.declare_function "down_8" pitch_to_pitch the_module
in

let struct_decls =
  let struct_decl m sdecl =
    let name = sdecl.sstruct_name
    and member_types = Array.of_list (List.map (fun (t,_,_) ->
        ltype_of_primitive t) sdecl.smembers)
    in let stype = L.struct_type context member_types in
    StringMap.add name (stype, sdecl.smembers) m in
  List.fold_left struct_decl StringMap.empty structs
in
let struct_lookup n = try StringMap.find n struct_decls
  with Not_found -> raise (Failure ("struct " ^ n ^ " not found")) in
let ltype_of_typ = function
  A.Struct n -> fst (struct_lookup n)
  | t -> ltype_of_primitive t
in
(* Define each function (arguments and return type) so we can
   call it even before we've created its body *)
let function_decls : (L.llvalue * sfunc_decl) StringMap.t =
  let function_decl m fdecl =
    let name = fdecl.sfname
    and formal_types =
     Array.of_list (List.map (fun (t,_,_) -> ltype_of_typ t) fdecl.sformals)
    in let ftype = L.function_type (ltype_of_typ fdecl.styp) formal_types in
  StringMap.add name (L.define_function name ftype the_module, fdecl) m in
List.fold_left function_decl StringMap.empty functions in

(* Fill in the body of the given function *)
let build_function_body fdecl =
```

```
let (the_function, _) = StringMap.find fdecl.sfname function_decls in
let builder = L.builder_at_end context (L.entry_block the_function) in
let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder
and float_format_str = L.build_global_stringptr "%g\n" "fmt" builder 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

  (* Allocate space for any locally declared variables and add the
   * resulting registers to our map *)
  and add_local m (t, n, _) =

  let local_var = L.build_alloca (ltype_of_typ t) n builder in
  StringMap.add n local_var m in
  let formals = List.fold_left2 add_formal StringMap.empty fdecl.sformals
  (Array.to_list (L.params the_function)) in
List.fold_left add_local formals fdecl.slocals
in
(* Return the value for a variable or formal argument.
   Check local names first, then global names *)
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 ((_, e) : sexpr) = match e with
    SLiteral i -> L.const_int i32_t i
  | SBoolLit b -> L.const_int i1_t (if b then 1 else 0)
  | SFliteral l -> L.const_float_of_string float_t l
  | SSliteral s -> L.build_global_stringptr s "123" builder
  | SArrayLit (sexpr_list) ->
```

```
    let all_elem = List.map (fun e ->
        expr builder e) sexpr_list in
    let llarray_t = L.type_of (List.hd all_elem) in
    let num_elems = List.length sexpr_list in
      let ptr = L.build_array_malloc llarray_t
          (L.const_int i32_t num_elems) "" builder
      in
      ignore (List.fold_left (fun i elem ->
          let idx = L.const_int i32_t i in
          let eptr = L.build_gep ptr [|idx|] "" builder in
          let cptr = L.build_pointercast eptr
              (L.pointer_type (L.type_of elem)) "" builder in
          let _ = (L.build_store elem cptr builder)
          in i+1)
          0 all_elem); ptr
  | SStructAccess(s, m) ->
    let (stype,location) = match s with
    (t,SId id) -> (t, lookup id)
    | _ -> raise (Failure("Illegal struct-type."))
    in
    let members = snd (struct_lookup (A.string_of_typ stype)) in
      let rec get_idx n lst i = match lst with
          | [] -> raise (Failure("Id " ^ m ^ " is not a member of " ^
             string_of_sexpr s))
          | hd::tl -> if hd=n then i else get_idx n tl (i+1)
        in let idx = (get_idx m (List.map (fun (_,nm,_) -> nm) members) 0) in
        let ptr = L.build_struct_gep location idx ("struct.ptr") builder in
    L.build_load ptr ("struct.val."^m) builder

  | SArrayAccess(arr, i) ->
    let arr_var = expr builder arr in
    let idx = expr builder i in
    let ptr =
      L.build_load (L.build_gep arr_var
                    [| idx |] "" builder)
        "" builder in ptr
  | SPliteral p -> L.build_global_stringptr p "4#" builder
```

```
| SNoexpr    -> L.const_int i32_t 0
| SId s      -> L.build_load (lookup s) s builder


| SAssign ((_,SStructAccess(s,m)), e) ->
    let e' = expr builder e in
    let (stype,location) = match s with
    (t,SId id) -> (t, lookup id)
    | _ -> raise (Failure("Illegal struct-type."))
    in
      let members = snd (struct_lookup (A.string_of_typ stype)) in
        let rec get_idx n lst i = match lst with
          | [] -> raise (Failure("Id " ^ m ^ " is not a member of struct "
              ^ string_of_sexpr s))
          | hd::tl -> if (hd=n) then i else get_idx n tl (i+1)
        in
        let idx = get_idx m (List.map (fun (_,nm,_) -> nm) members) 0 in
      let ptr = L.build_struct_gep location idx "struct.ptr" builder in
    ignore(L.build_store e' ptr builder); e'



| SAssign ((_,SArrayAccess(arr, i)), e) ->
  let e' = expr builder e in
  let arr_var = expr builder arr in
  let idx = expr builder i in
  let ptr =
    L.build_gep arr_var [| idx |] "" builder
  in
  ignore(L.build_store e' ptr builder); e'
| SAssign (e1, e2) ->
let e2' = expr builder e2 in
(match snd e1 with
     SId s -> ignore(L.build_store e2' (lookup s) builder); e2'
    | _ -> raise (Failure ("Not implemented in codegen"))
)
| SBinop ((A.Array(t1, sz1), e1), A.Con, (A.Array(t2, sz2), e2))->
```

```
    let llarray_t = (ltype_of_primitive t1) in
    let ptr = L.build_array_malloc llarray_t
          (L.const_int i32_t (sz1+sz2)) "" builder in
    for i = 0 to sz1+sz2-1 do
        let arr_var = if i < sz1 then expr builder (A.Array(t1, sz1), e1)
                              else expr builder (A.Array(t2, sz2), e2) in
        let idx = if i < sz1 then L.const_int i32_t i
                  else L.const_int i32_t (i-sz1) in
        let access = L.build_load (L.build_gep arr_var [| idx |] "" builder)
            "temp" builder in
        let eptr = L.build_gep ptr [|L.const_int i32_t i |] "" builder in
        let cptr = L.build_pointercast eptr
            (L.pointer_type llarray_t) "" builder in
        ignore(L.build_store access cptr builder)
    done; ptr
  | SBinop ((A.Float,_) as e1, op, e2) ->
    let e1' = expr builder e1
    and e2' = expr builder e2 in
    (match op with
      A.Add     -> L.build_fadd
    | A.Sub     -> L.build_fsub
    | A.Mult    -> L.build_fmul
    | A.Div     -> L.build_fdiv
    | A.Mod     -> L.build_frem
    | A.Equal   -> L.build_fcmp L.Fcmp.Oeq
    | A.Neq     -> L.build_fcmp L.Fcmp.One
    | A.Less    -> L.build_fcmp L.Fcmp.Olt
    | A.Leq     -> L.build_fcmp L.Fcmp.Ole
    | A.Greater -> L.build_fcmp L.Fcmp.Ogt
    | A.Geq     -> L.build_fcmp L.Fcmp.Oge
    | A.And | A.Or ->
        raise (Failure "internal error: semant should have rejected and/or
            on float")
    | _ -> raise (Failure("invalid binary operator"^A.string_of_op op))
    ) e1' e2' "tmp" builder
  | SBinop (e1, op, e2) ->
    let e1' = expr builder e1
```

```
    and e2' = expr builder e2 in
    (match op with
      A.Add     -> L.build_add
    | A.Sub     -> L.build_sub
    | A.Mult    -> L.build_mul
    | A.Div     -> L.build_sdiv
    | A.Mod     -> 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("invalid binary operator"^A.string_of_op op))
    ) e1' e2' "tmp" builder
  | SUnop(op, ((t, _) as e)) ->
      let e' = expr builder e in
(match op with
  A.Neg when t = A.Float -> L.build_fneg
| A.Neg                  -> L.build_neg
      | A.Not                  -> L.build_not) e' "tmp" builder
| SCall ("print", [t,e]) | SCall ("printb", [t,e]) ->
  (match t with
  A.Int -> L.build_call printf_func [| int_format_str ; (expr builder
      (t,e)) |] "printf" builder
  | A.String -> L.build_call prints_func [| (expr builder (t,e)) |]
      "prints" builder
  | A.Float -> L.build_call printf_func [| float_format_str ; (expr builder
      (t,e)) |] "printf" builder
  | A.Pitch -> L.build_call prints_func [|(expr builder (t,e)) |] "prints"
      builder
  | _ -> raise (Failure (A.string_of_typ t)))
| SCall ("size",[t,_]) -> (match t with
  A.Array (_,b) -> L.const_int i32_t b
  | _ -> raise (Failure("Array not found")))
```

```
| SCall ("pitch_to_int", e) -> L.build_call pitch_to_int_func [|expr
    builder (List.hd e)|] "pitchtoint" builder
| SCall ("random", e) -> L.build_call random_func [|expr builder (List.hd
    e)|] "ran" builder
| SCall ("up_8", e) -> L.build_call up8_func [|expr builder (List.hd e)|]
    "" builder
| SCall ("down_8", e) -> L.build_call down8_func [|expr builder (List.hd
    e)|] "" builder


| SCall (f, args) ->
   let (fdef, fdecl) = StringMap.find f function_decls in
   let llargs = List.rev (List.map (expr builder) (List.rev args)) in
   let result = (match fdecl.styp with
                 A.Void -> ""
               | _ -> f ^ "_result") in
   L.build_call fdef (Array.of_list llargs) result builder
in

(* LLVM insists each basic block end with exactly one "terminator"
   instruction that transfers control. This function runs "instr builder"
   if the current block does not already have a terminator. Used,
   e.g., to handle the "fall off the end of the function" case. *)
let add_terminal builder instr =
  match L.block_terminator (L.insertion_block builder) with
  Some _ -> ()
  | None -> ignore (instr builder) in

(* Build the code for the given statement; return the builder for
   the statement's successor (i.e., the next instruction will be built
   after the one generated by this call) *)

let rec stmt builder = function
    SBlock sl -> List.fold_left stmt builder sl
  | SExpr e -> ignore(expr builder e); builder
  | SReturn e -> ignore(match fdecl.styp with
                        (* Special "return nothing" instr *)
```

```
                        A.Void -> L.build_ret_void builder
                        (* Build return statement *)
                      | _ -> L.build_ret (expr builder e) builder );
               builder
  | SIf (predicate, then_stmt, else_stmt) ->
     let bool_val = expr builder predicate in
let merge_bb = L.append_block context "merge" the_function in
     let build_br_merge = L.build_br merge_bb in (* partial function *)

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

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

ignore(L.build_cond_br bool_val then_bb else_bb builder);
L.builder_at_end context merge_bb

  | SWhile (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

  (* Implement for loops as while loops *)
  | SFor (e1, e2, e3, body) -> stmt builder
```

```
      ( SBlock [SExpr e1 ; SWhile (e2, SBlock [body ; SExpr e3]) ] )
    in

    (* Build the code for each statement in the function *)
    let builder = stmt builder (SBlock fdecl.sbody) in

    (* Add a return if the last block falls off the end *)
    add_terminal builder (match fdecl.styp with
        A.Void -> L.build_ret_void
      | A.Float -> L.build_ret (L.const_float float_t 0.0)
      | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
  in

  List.iter build_function_body functions;
  the_module
```

8.7 midi_generator.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
// Author: Yuanji Huang

struct _track{
   int instrument;
   char** melody;
   int* rhythm;
};

char* up_8(char* s){
    return s;
}

char* down_8(char* s){
    return s;
```

```
}

int pitch_to_int(char* s){
    if(s[0] == '1'&&s[1] == 'b'&&s[2] == '1')
        return 0;
    char a = s[0];
    char b = s[1];
    char c = s[2];
    int result = 23;
    if(a-'0' < 4){
        result += 2 * (a-'0') - 1;
    }else{
        result += 2 * (a-'0') - 2;
    }
    switch(b){
        case 'b':
            result--;
            break;
        case '#':
            result++;
            break;
    }
    result += (c-'1') * 12;
    return result;
}

int ran(int range){
    return (rand() % range);
}
```

8.8 Sample generated llvm code of file include_struct_test.ll

```
; ModuleID = 'CAML'
source_filename = "CAML"

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
```

```
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
@"123" = private unnamed_addr constant [12 x i8] c"track_start\00", align 1
@fmt.2 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.3 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
@"4#" = private unnamed_addr constant [4 x i8] c"1b1\00", align 1
@fmt.4 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.5 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
@"4#.6" = private unnamed_addr constant [4 x i8] c"1^4\00", align 1

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

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

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

declare i32 @ran(i32, ...)

declare i32 @up_8(i8*, ...)

declare i32 @down_8(i8*, ...)

define void @generate_music({ i32*, i8**, i32, i32 } %t) {
entry:
  %t1 = alloca { i32*, i8**, i32, i32 }
  store { i32*, i8**, i32, i32 } %t, { i32*, i8**, i32, i32 }* %t1
  %i = alloca i32
  %prints = call i32 (i8*, ...) @puts(i8* getelementptr inbounds ([12 x i8],
      [12 x i8]* @"123", i32 0, i32 0))
  %struct.ptr = getelementptr inbounds { i32*, i8**, i32, i32 }, { i32*, i8**,
      i32, i32 }* %t1, i32 0, i32 3
  %struct.val.instrument = load i32, i32* %struct.ptr
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8],
      [4 x i8]* @fmt, i32 0, i32 0), i32 %struct.val.instrument)
  %struct.ptr2 = getelementptr inbounds { i32*, i8**, i32, i32 }, { i32*, i8**,
      i32, i32 }* %t1, i32 0, i32 2
  %struct.val.size = load i32, i32* %struct.ptr2
```

```
  %printf3 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8],
      [4 x i8]* @fmt, i32 0, i32 0), i32 %struct.val.size)
  store i32 0, i32* %i
  br label %while

while:                                            ; preds = %while_body, %entry
  %i11 = load i32, i32* %i
  %struct.ptr12 = getelementptr inbounds { i32*, i8**, i32, i32 }, { i32*,
      i8**, i32, i32 }* %t1, i32 0, i32 2
  %struct.val.size13 = load i32, i32* %struct.ptr12
  %tmp14 = icmp slt i32 %i11, %struct.val.size13
  br i1 %tmp14, label %while_body, label %merge

while_body:                                       ; preds = %while
  %struct.ptr4 = getelementptr inbounds { i32*, i8**, i32, i32 }, { i32*, i8**,
      i32, i32 }* %t1, i32 0, i32 1
  %struct.val.melody = load i8**, i8*** %struct.ptr4
  %i5 = load i32, i32* %i
  %0 = getelementptr i8*, i8** %struct.val.melody, i32 %i5
  %1 = load i8*, i8** %0
  %pitchtoint = call i32 (i8*, ...) @pitch_to_int(i8* %1)
  %printf6 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8],
      [4 x i8]* @fmt, i32 0, i32 0), i32 %pitchtoint)
  %struct.ptr7 = getelementptr inbounds { i32*, i8**, i32, i32 }, { i32*, i8**,
      i32, i32 }* %t1, i32 0, i32 0
  %struct.val.rhythm = load i32*, i32** %struct.ptr7
  %i8 = load i32, i32* %i
  %2 = getelementptr i32, i32* %struct.val.rhythm, i32 %i8
  %3 = load i32, i32* %2
  %printf9 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8],
      [4 x i8]* @fmt, i32 0, i32 0), i32 %3)
  %i10 = load i32, i32* %i
  %tmp = add i32 %i10, 1
  store i32 %tmp, i32* %i
  br label %while

merge:                                            ; preds = %while
```

```
  ret void
}

define { i32*, i8**, i32, i32 } @build_track(i32 %instrument, i32 %size, i8**
    %melody, i32* %rhythm) {
entry:
  %instrument1 = alloca i32
  store i32 %instrument, i32* %instrument1
  %size2 = alloca i32
  store i32 %size, i32* %size2
  %melody3 = alloca i8**
  store i8** %melody, i8*** %melody3
  %rhythm4 = alloca i32*
  store i32* %rhythm, i32** %rhythm4
  %res = alloca { i32*, i8**, i32, i32 }
  %i = alloca i32
  %new_melody = alloca i8**
  %new_rhythm = alloca i32*
  %melody5 = load i8**, i8*** %melody3
  %0 = getelementptr i8*, i8** %melody5, i32 0
  %1 = load i8*, i8** %0
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
      i1** null, i32 1) to i32))
  %2 = bitcast i8* %malloccall to i8**
  %3 = getelementptr i8*, i8** %2, i32 0
  store i8* %1, i8** %3
  store i8** %2, i8*** %new_melody
  %rhythm6 = load i32*, i32** %rhythm4
  %4 = getelementptr i32, i32* %rhythm6, i32 0
  %5 = load i32, i32* %4
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32,
      i32* null, i32 1) to i32))
  %6 = bitcast i8* %malloccall7 to i32*
  %7 = getelementptr i32, i32* %6, i32 0
  store i32 %5, i32* %7
  store i32* %6, i32** %new_rhythm
  store i32 1, i32* %i
```

```
  br label %while

while:                                            ; preds = %while_body, %entry
  %i222 = load i32, i32* %i
  %size223 = load i32, i32* %size2
  %tmp224 = icmp slt i32 %i222, %size223
  br i1 %tmp224, label %while_body, label %merge

while_body:                                       ; preds = %while
  %malloccall8 = tail call i8* @malloc(i32 mul (i32 ptrtoint (i1**
      getelementptr (i1*, i1** null, i32 1) to i32), i32 51))
  %8 = bitcast i8* %malloccall8 to i8**
  %new_melody9 = load i8**, i8*** %new_melody
  %9 = getelementptr i8*, i8** %new_melody9, i32 0
  %temp = load i8*, i8** %9
  %10 = getelementptr i8*, i8** %8, i32 0
  store i8* %temp, i8** %10
  %new_melody10 = load i8**, i8*** %new_melody
  %11 = getelementptr i8*, i8** %new_melody10, i32 1
  %temp11 = load i8*, i8** %11
  %12 = getelementptr i8*, i8** %8, i32 1
  store i8* %temp11, i8** %12
  %new_melody12 = load i8**, i8*** %new_melody
  %13 = getelementptr i8*, i8** %new_melody12, i32 2
  %temp13 = load i8*, i8** %13
  %14 = getelementptr i8*, i8** %8, i32 2
  store i8* %temp13, i8** %14
  %new_melody14 = load i8**, i8*** %new_melody
  %15 = getelementptr i8*, i8** %new_melody14, i32 3
  %temp15 = load i8*, i8** %15
  %16 = getelementptr i8*, i8** %8, i32 3
  store i8* %temp15, i8** %16
  %new_melody16 = load i8**, i8*** %new_melody
  %17 = getelementptr i8*, i8** %new_melody16, i32 4
  %temp17 = load i8*, i8** %17
  %18 = getelementptr i8*, i8** %8, i32 4
  store i8* %temp17, i8** %18
```

```
%new_melody18 = load i8**, i8*** %new_melody
%19 = getelementptr i8*, i8** %new_melody18, i32 5
%temp19 = load i8*, i8** %19
%20 = getelementptr i8*, i8** %8, i32 5
store i8* %temp19, i8** %20
%new_melody20 = load i8**, i8*** %new_melody
%21 = getelementptr i8*, i8** %new_melody20, i32 6
%temp21 = load i8*, i8** %21
%22 = getelementptr i8*, i8** %8, i32 6
store i8* %temp21, i8** %22
%new_melody22 = load i8**, i8*** %new_melody
%23 = getelementptr i8*, i8** %new_melody22, i32 7
%temp23 = load i8*, i8** %23
%24 = getelementptr i8*, i8** %8, i32 7
store i8* %temp23, i8** %24
%new_melody24 = load i8**, i8*** %new_melody
%25 = getelementptr i8*, i8** %new_melody24, i32 8
%temp25 = load i8*, i8** %25
%26 = getelementptr i8*, i8** %8, i32 8
store i8* %temp25, i8** %26
%new_melody26 = load i8**, i8*** %new_melody
%27 = getelementptr i8*, i8** %new_melody26, i32 9
%temp27 = load i8*, i8** %27
%28 = getelementptr i8*, i8** %8, i32 9
store i8* %temp27, i8** %28
%new_melody28 = load i8**, i8*** %new_melody
%29 = getelementptr i8*, i8** %new_melody28, i32 10
%temp29 = load i8*, i8** %29
%30 = getelementptr i8*, i8** %8, i32 10
store i8* %temp29, i8** %30
%new_melody30 = load i8**, i8*** %new_melody
%31 = getelementptr i8*, i8** %new_melody30, i32 11
%temp31 = load i8*, i8** %31
%32 = getelementptr i8*, i8** %8, i32 11
store i8* %temp31, i8** %32
%new_melody32 = load i8**, i8*** %new_melody
%33 = getelementptr i8*, i8** %new_melody32, i32 12
```

```
%temp33 = load i8*, i8** %33
%34 = getelementptr i8*, i8** %8, i32 12
store i8* %temp33, i8** %34
%new_melody34 = load i8**, i8*** %new_melody
%35 = getelementptr i8*, i8** %new_melody34, i32 13
%temp35 = load i8*, i8** %35
%36 = getelementptr i8*, i8** %8, i32 13
store i8* %temp35, i8** %36
%new_melody36 = load i8**, i8*** %new_melody
%37 = getelementptr i8*, i8** %new_melody36, i32 14
%temp37 = load i8*, i8** %37
%38 = getelementptr i8*, i8** %8, i32 14
store i8* %temp37, i8** %38
%new_melody38 = load i8**, i8*** %new_melody
%39 = getelementptr i8*, i8** %new_melody38, i32 15
%temp39 = load i8*, i8** %39
%40 = getelementptr i8*, i8** %8, i32 15
store i8* %temp39, i8** %40
%new_melody40 = load i8**, i8*** %new_melody
%41 = getelementptr i8*, i8** %new_melody40, i32 16
%temp41 = load i8*, i8** %41
%42 = getelementptr i8*, i8** %8, i32 16
store i8* %temp41, i8** %42
%new_melody42 = load i8**, i8*** %new_melody
%43 = getelementptr i8*, i8** %new_melody42, i32 17
%temp43 = load i8*, i8** %43
%44 = getelementptr i8*, i8** %8, i32 17
store i8* %temp43, i8** %44
%new_melody44 = load i8**, i8*** %new_melody
%45 = getelementptr i8*, i8** %new_melody44, i32 18
%temp45 = load i8*, i8** %45
%46 = getelementptr i8*, i8** %8, i32 18
store i8* %temp45, i8** %46
%new_melody46 = load i8**, i8*** %new_melody
%47 = getelementptr i8*, i8** %new_melody46, i32 19
%temp47 = load i8*, i8** %47
%48 = getelementptr i8*, i8** %8, i32 19
```

```
store i8* %temp47, i8** %48
%new_melody48 = load i8**, i8*** %new_melody
%49 = getelementptr i8*, i8** %new_melody48, i32 20
%temp49 = load i8*, i8** %49
%50 = getelementptr i8*, i8** %8, i32 20
store i8* %temp49, i8** %50
%new_melody50 = load i8**, i8*** %new_melody
%51 = getelementptr i8*, i8** %new_melody50, i32 21
%temp51 = load i8*, i8** %51
%52 = getelementptr i8*, i8** %8, i32 21
store i8* %temp51, i8** %52
%new_melody52 = load i8**, i8*** %new_melody
%53 = getelementptr i8*, i8** %new_melody52, i32 22
%temp53 = load i8*, i8** %53
%54 = getelementptr i8*, i8** %8, i32 22
store i8* %temp53, i8** %54
%new_melody54 = load i8**, i8*** %new_melody
%55 = getelementptr i8*, i8** %new_melody54, i32 23
%temp55 = load i8*, i8** %55
%56 = getelementptr i8*, i8** %8, i32 23
store i8* %temp55, i8** %56
%new_melody56 = load i8**, i8*** %new_melody
%57 = getelementptr i8*, i8** %new_melody56, i32 24
%temp57 = load i8*, i8** %57
%58 = getelementptr i8*, i8** %8, i32 24
store i8* %temp57, i8** %58
%new_melody58 = load i8**, i8*** %new_melody
%59 = getelementptr i8*, i8** %new_melody58, i32 25
%temp59 = load i8*, i8** %59
%60 = getelementptr i8*, i8** %8, i32 25
store i8* %temp59, i8** %60
%new_melody60 = load i8**, i8*** %new_melody
%61 = getelementptr i8*, i8** %new_melody60, i32 26
%temp61 = load i8*, i8** %61
%62 = getelementptr i8*, i8** %8, i32 26
store i8* %temp61, i8** %62
%new_melody62 = load i8**, i8*** %new_melody
```

```
%63 = getelementptr i8*, i8** %new_melody62, i32 27
%temp63 = load i8*, i8** %63
%64 = getelementptr i8*, i8** %8, i32 27
store i8* %temp63, i8** %64
%new_melody64 = load i8**, i8*** %new_melody
%65 = getelementptr i8*, i8** %new_melody64, i32 28
%temp65 = load i8*, i8** %65
%66 = getelementptr i8*, i8** %8, i32 28
store i8* %temp65, i8** %66
%new_melody66 = load i8**, i8*** %new_melody
%67 = getelementptr i8*, i8** %new_melody66, i32 29
%temp67 = load i8*, i8** %67
%68 = getelementptr i8*, i8** %8, i32 29
store i8* %temp67, i8** %68
%new_melody68 = load i8**, i8*** %new_melody
%69 = getelementptr i8*, i8** %new_melody68, i32 30
%temp69 = load i8*, i8** %69
%70 = getelementptr i8*, i8** %8, i32 30
store i8* %temp69, i8** %70
%new_melody70 = load i8**, i8*** %new_melody
%71 = getelementptr i8*, i8** %new_melody70, i32 31
%temp71 = load i8*, i8** %71
%72 = getelementptr i8*, i8** %8, i32 31
store i8* %temp71, i8** %72
%new_melody72 = load i8**, i8*** %new_melody
%73 = getelementptr i8*, i8** %new_melody72, i32 32
%temp73 = load i8*, i8** %73
%74 = getelementptr i8*, i8** %8, i32 32
store i8* %temp73, i8** %74
%new_melody74 = load i8**, i8*** %new_melody
%75 = getelementptr i8*, i8** %new_melody74, i32 33
%temp75 = load i8*, i8** %75
%76 = getelementptr i8*, i8** %8, i32 33
store i8* %temp75, i8** %76
%new_melody76 = load i8**, i8*** %new_melody
%77 = getelementptr i8*, i8** %new_melody76, i32 34
%temp77 = load i8*, i8** %77
```

```
%78 = getelementptr i8*, i8** %8, i32 34
store i8* %temp77, i8** %78
%new_melody78 = load i8**, i8*** %new_melody
%79 = getelementptr i8*, i8** %new_melody78, i32 35
%temp79 = load i8*, i8** %79
%80 = getelementptr i8*, i8** %8, i32 35
store i8* %temp79, i8** %80
%new_melody80 = load i8**, i8*** %new_melody
%81 = getelementptr i8*, i8** %new_melody80, i32 36
%temp81 = load i8*, i8** %81
%82 = getelementptr i8*, i8** %8, i32 36
store i8* %temp81, i8** %82
%new_melody82 = load i8**, i8*** %new_melody
%83 = getelementptr i8*, i8** %new_melody82, i32 37
%temp83 = load i8*, i8** %83
%84 = getelementptr i8*, i8** %8, i32 37
store i8* %temp83, i8** %84
%new_melody84 = load i8**, i8*** %new_melody
%85 = getelementptr i8*, i8** %new_melody84, i32 38
%temp85 = load i8*, i8** %85
%86 = getelementptr i8*, i8** %8, i32 38
store i8* %temp85, i8** %86
%new_melody86 = load i8**, i8*** %new_melody
%87 = getelementptr i8*, i8** %new_melody86, i32 39
%temp87 = load i8*, i8** %87
%88 = getelementptr i8*, i8** %8, i32 39
store i8* %temp87, i8** %88
%new_melody88 = load i8**, i8*** %new_melody
%89 = getelementptr i8*, i8** %new_melody88, i32 40
%temp89 = load i8*, i8** %89
%90 = getelementptr i8*, i8** %8, i32 40
store i8* %temp89, i8** %90
%new_melody90 = load i8**, i8*** %new_melody
%91 = getelementptr i8*, i8** %new_melody90, i32 41
%temp91 = load i8*, i8** %91
%92 = getelementptr i8*, i8** %8, i32 41
store i8* %temp91, i8** %92
```

```
%new_melody92 = load i8**, i8*** %new_melody
%93 = getelementptr i8*, i8** %new_melody92, i32 42
%temp93 = load i8*, i8** %93
%94 = getelementptr i8*, i8** %8, i32 42
store i8* %temp93, i8** %94
%new_melody94 = load i8**, i8*** %new_melody
%95 = getelementptr i8*, i8** %new_melody94, i32 43
%temp95 = load i8*, i8** %95
%96 = getelementptr i8*, i8** %8, i32 43
store i8* %temp95, i8** %96
%new_melody96 = load i8**, i8*** %new_melody
%97 = getelementptr i8*, i8** %new_melody96, i32 44
%temp97 = load i8*, i8** %97
%98 = getelementptr i8*, i8** %8, i32 44
store i8* %temp97, i8** %98
%new_melody98 = load i8**, i8*** %new_melody
%99 = getelementptr i8*, i8** %new_melody98, i32 45
%temp99 = load i8*, i8** %99
%100 = getelementptr i8*, i8** %8, i32 45
store i8* %temp99, i8** %100
%new_melody100 = load i8**, i8*** %new_melody
%101 = getelementptr i8*, i8** %new_melody100, i32 46
%temp101 = load i8*, i8** %101
%102 = getelementptr i8*, i8** %8, i32 46
store i8* %temp101, i8** %102
%new_melody102 = load i8**, i8*** %new_melody
%103 = getelementptr i8*, i8** %new_melody102, i32 47
%temp103 = load i8*, i8** %103
%104 = getelementptr i8*, i8** %8, i32 47
store i8* %temp103, i8** %104
%new_melody104 = load i8**, i8*** %new_melody
%105 = getelementptr i8*, i8** %new_melody104, i32 48
%temp105 = load i8*, i8** %105
%106 = getelementptr i8*, i8** %8, i32 48
store i8* %temp105, i8** %106
%new_melody106 = load i8**, i8*** %new_melody
%107 = getelementptr i8*, i8** %new_melody106, i32 49
```

```
%temp107 = load i8*, i8** %107
%108 = getelementptr i8*, i8** %8, i32 49
store i8* %temp107, i8** %108
%malloccall108 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i32))
%109 = bitcast i8* %malloccall108 to i8**
%110 = getelementptr i8*, i8** %109, i32 0
store i8* getelementptr inbounds ([4 x i8], [4 x i8]* @"4#", i32 0, i32 0),
    i8** %110
%111 = getelementptr i8*, i8** %109, i32 0
%temp109 = load i8*, i8** %111
%112 = getelementptr i8*, i8** %8, i32 50
store i8* %temp109, i8** %112
store i8** %8, i8*** %new_melody
%melody110 = load i8**, i8*** %melody3
%i111 = load i32, i32* %i
%113 = getelementptr i8*, i8** %melody110, i32 %i111
%114 = load i8*, i8** %113
%new_melody112 = load i8**, i8*** %new_melody
%i113 = load i32, i32* %i
%115 = getelementptr i8*, i8** %new_melody112, i32 %i113
store i8* %114, i8** %115
%malloccall114 = tail call i8* @malloc(i32 mul (i32 ptrtoint (i32*
    getelementptr (i32, i32* null, i32 1) to i32), i32 51))
%116 = bitcast i8* %malloccall114 to i32*
%new_rhythm115 = load i32*, i32** %new_rhythm
%117 = getelementptr i32, i32* %new_rhythm115, i32 0
%temp116 = load i32, i32* %117
%118 = getelementptr i32, i32* %116, i32 0
store i32 %temp116, i32* %118
%new_rhythm117 = load i32*, i32** %new_rhythm
%119 = getelementptr i32, i32* %new_rhythm117, i32 1
%temp118 = load i32, i32* %119
%120 = getelementptr i32, i32* %116, i32 1
store i32 %temp118, i32* %120
%new_rhythm119 = load i32*, i32** %new_rhythm
%121 = getelementptr i32, i32* %new_rhythm119, i32 2
```

```
%temp120 = load i32, i32* %121
%122 = getelementptr i32, i32* %116, i32 2
store i32 %temp120, i32* %122
%new_rhythm121 = load i32*, i32** %new_rhythm
%123 = getelementptr i32, i32* %new_rhythm121, i32 3
%temp122 = load i32, i32* %123
%124 = getelementptr i32, i32* %116, i32 3
store i32 %temp122, i32* %124
%new_rhythm123 = load i32*, i32** %new_rhythm
%125 = getelementptr i32, i32* %new_rhythm123, i32 4
%temp124 = load i32, i32* %125
%126 = getelementptr i32, i32* %116, i32 4
store i32 %temp124, i32* %126
%new_rhythm125 = load i32*, i32** %new_rhythm
%127 = getelementptr i32, i32* %new_rhythm125, i32 5
%temp126 = load i32, i32* %127
%128 = getelementptr i32, i32* %116, i32 5
store i32 %temp126, i32* %128
%new_rhythm127 = load i32*, i32** %new_rhythm
%129 = getelementptr i32, i32* %new_rhythm127, i32 6
%temp128 = load i32, i32* %129
%130 = getelementptr i32, i32* %116, i32 6
store i32 %temp128, i32* %130
%new_rhythm129 = load i32*, i32** %new_rhythm
%131 = getelementptr i32, i32* %new_rhythm129, i32 7
%temp130 = load i32, i32* %131
%132 = getelementptr i32, i32* %116, i32 7
store i32 %temp130, i32* %132
%new_rhythm131 = load i32*, i32** %new_rhythm
%133 = getelementptr i32, i32* %new_rhythm131, i32 8
%temp132 = load i32, i32* %133
%134 = getelementptr i32, i32* %116, i32 8
store i32 %temp132, i32* %134
%new_rhythm133 = load i32*, i32** %new_rhythm
%135 = getelementptr i32, i32* %new_rhythm133, i32 9
%temp134 = load i32, i32* %135
%136 = getelementptr i32, i32* %116, i32 9
```

```
store i32 %temp134, i32* %136
%new_rhythm135 = load i32*, i32** %new_rhythm
%137 = getelementptr i32, i32* %new_rhythm135, i32 10
%temp136 = load i32, i32* %137
%138 = getelementptr i32, i32* %116, i32 10
store i32 %temp136, i32* %138
%new_rhythm137 = load i32*, i32** %new_rhythm
%139 = getelementptr i32, i32* %new_rhythm137, i32 11
%temp138 = load i32, i32* %139
%140 = getelementptr i32, i32* %116, i32 11
store i32 %temp138, i32* %140
%new_rhythm139 = load i32*, i32** %new_rhythm
%141 = getelementptr i32, i32* %new_rhythm139, i32 12
%temp140 = load i32, i32* %141
%142 = getelementptr i32, i32* %116, i32 12
store i32 %temp140, i32* %142
%new_rhythm141 = load i32*, i32** %new_rhythm
%143 = getelementptr i32, i32* %new_rhythm141, i32 13
%temp142 = load i32, i32* %143
%144 = getelementptr i32, i32* %116, i32 13
store i32 %temp142, i32* %144
%new_rhythm143 = load i32*, i32** %new_rhythm
%145 = getelementptr i32, i32* %new_rhythm143, i32 14
%temp144 = load i32, i32* %145
%146 = getelementptr i32, i32* %116, i32 14
store i32 %temp144, i32* %146
%new_rhythm145 = load i32*, i32** %new_rhythm
%147 = getelementptr i32, i32* %new_rhythm145, i32 15
%temp146 = load i32, i32* %147
%148 = getelementptr i32, i32* %116, i32 15
store i32 %temp146, i32* %148
%new_rhythm147 = load i32*, i32** %new_rhythm
%149 = getelementptr i32, i32* %new_rhythm147, i32 16
%temp148 = load i32, i32* %149
%150 = getelementptr i32, i32* %116, i32 16
store i32 %temp148, i32* %150
%new_rhythm149 = load i32*, i32** %new_rhythm
```

```
%151 = getelementptr i32, i32* %new_rhythm149, i32 17
%temp150 = load i32, i32* %151
%152 = getelementptr i32, i32* %116, i32 17
store i32 %temp150, i32* %152
%new_rhythm151 = load i32*, i32** %new_rhythm
%153 = getelementptr i32, i32* %new_rhythm151, i32 18
%temp152 = load i32, i32* %153
%154 = getelementptr i32, i32* %116, i32 18
store i32 %temp152, i32* %154
%new_rhythm153 = load i32*, i32** %new_rhythm
%155 = getelementptr i32, i32* %new_rhythm153, i32 19
%temp154 = load i32, i32* %155
%156 = getelementptr i32, i32* %116, i32 19
store i32 %temp154, i32* %156
%new_rhythm155 = load i32*, i32** %new_rhythm
%157 = getelementptr i32, i32* %new_rhythm155, i32 20
%temp156 = load i32, i32* %157
%158 = getelementptr i32, i32* %116, i32 20
store i32 %temp156, i32* %158
%new_rhythm157 = load i32*, i32** %new_rhythm
%159 = getelementptr i32, i32* %new_rhythm157, i32 21
%temp158 = load i32, i32* %159
%160 = getelementptr i32, i32* %116, i32 21
store i32 %temp158, i32* %160
%new_rhythm159 = load i32*, i32** %new_rhythm
%161 = getelementptr i32, i32* %new_rhythm159, i32 22
%temp160 = load i32, i32* %161
%162 = getelementptr i32, i32* %116, i32 22
store i32 %temp160, i32* %162
%new_rhythm161 = load i32*, i32** %new_rhythm
%163 = getelementptr i32, i32* %new_rhythm161, i32 23
%temp162 = load i32, i32* %163
%164 = getelementptr i32, i32* %116, i32 23
store i32 %temp162, i32* %164
%new_rhythm163 = load i32*, i32** %new_rhythm
%165 = getelementptr i32, i32* %new_rhythm163, i32 24
%temp164 = load i32, i32* %165
```

```
%166 = getelementptr i32, i32* %116, i32 24
store i32 %temp164, i32* %166
%new_rhythm165 = load i32*, i32** %new_rhythm
%167 = getelementptr i32, i32* %new_rhythm165, i32 25
%temp166 = load i32, i32* %167
%168 = getelementptr i32, i32* %116, i32 25
store i32 %temp166, i32* %168
%new_rhythm167 = load i32*, i32** %new_rhythm
%169 = getelementptr i32, i32* %new_rhythm167, i32 26
%temp168 = load i32, i32* %169
%170 = getelementptr i32, i32* %116, i32 26
store i32 %temp168, i32* %170
%new_rhythm169 = load i32*, i32** %new_rhythm
%171 = getelementptr i32, i32* %new_rhythm169, i32 27
%temp170 = load i32, i32* %171
%172 = getelementptr i32, i32* %116, i32 27
store i32 %temp170, i32* %172
%new_rhythm171 = load i32*, i32** %new_rhythm
%173 = getelementptr i32, i32* %new_rhythm171, i32 28
%temp172 = load i32, i32* %173
%174 = getelementptr i32, i32* %116, i32 28
store i32 %temp172, i32* %174
%new_rhythm173 = load i32*, i32** %new_rhythm
%175 = getelementptr i32, i32* %new_rhythm173, i32 29
%temp174 = load i32, i32* %175
%176 = getelementptr i32, i32* %116, i32 29
store i32 %temp174, i32* %176
%new_rhythm175 = load i32*, i32** %new_rhythm
%177 = getelementptr i32, i32* %new_rhythm175, i32 30
%temp176 = load i32, i32* %177
%178 = getelementptr i32, i32* %116, i32 30
store i32 %temp176, i32* %178
%new_rhythm177 = load i32*, i32** %new_rhythm
%179 = getelementptr i32, i32* %new_rhythm177, i32 31
%temp178 = load i32, i32* %179
%180 = getelementptr i32, i32* %116, i32 31
store i32 %temp178, i32* %180
```

```
%new_rhythm179 = load i32*, i32** %new_rhythm
%181 = getelementptr i32, i32* %new_rhythm179, i32 32
%temp180 = load i32, i32* %181
%182 = getelementptr i32, i32* %116, i32 32
store i32 %temp180, i32* %182
%new_rhythm181 = load i32*, i32** %new_rhythm
%183 = getelementptr i32, i32* %new_rhythm181, i32 33
%temp182 = load i32, i32* %183
%184 = getelementptr i32, i32* %116, i32 33
store i32 %temp182, i32* %184
%new_rhythm183 = load i32*, i32** %new_rhythm
%185 = getelementptr i32, i32* %new_rhythm183, i32 34
%temp184 = load i32, i32* %185
%186 = getelementptr i32, i32* %116, i32 34
store i32 %temp184, i32* %186
%new_rhythm185 = load i32*, i32** %new_rhythm
%187 = getelementptr i32, i32* %new_rhythm185, i32 35
%temp186 = load i32, i32* %187
%188 = getelementptr i32, i32* %116, i32 35
store i32 %temp186, i32* %188
%new_rhythm187 = load i32*, i32** %new_rhythm
%189 = getelementptr i32, i32* %new_rhythm187, i32 36
%temp188 = load i32, i32* %189
%190 = getelementptr i32, i32* %116, i32 36
store i32 %temp188, i32* %190
%new_rhythm189 = load i32*, i32** %new_rhythm
%191 = getelementptr i32, i32* %new_rhythm189, i32 37
%temp190 = load i32, i32* %191
%192 = getelementptr i32, i32* %116, i32 37
store i32 %temp190, i32* %192
%new_rhythm191 = load i32*, i32** %new_rhythm
%193 = getelementptr i32, i32* %new_rhythm191, i32 38
%temp192 = load i32, i32* %193
%194 = getelementptr i32, i32* %116, i32 38
store i32 %temp192, i32* %194
%new_rhythm193 = load i32*, i32** %new_rhythm
%195 = getelementptr i32, i32* %new_rhythm193, i32 39
```

```
%temp194 = load i32, i32* %195
%196 = getelementptr i32, i32* %116, i32 39
store i32 %temp194, i32* %196
%new_rhythm195 = load i32*, i32** %new_rhythm
%197 = getelementptr i32, i32* %new_rhythm195, i32 40
%temp196 = load i32, i32* %197
%198 = getelementptr i32, i32* %116, i32 40
store i32 %temp196, i32* %198
%new_rhythm197 = load i32*, i32** %new_rhythm
%199 = getelementptr i32, i32* %new_rhythm197, i32 41
%temp198 = load i32, i32* %199
%200 = getelementptr i32, i32* %116, i32 41
store i32 %temp198, i32* %200
%new_rhythm199 = load i32*, i32** %new_rhythm
%201 = getelementptr i32, i32* %new_rhythm199, i32 42
%temp200 = load i32, i32* %201
%202 = getelementptr i32, i32* %116, i32 42
store i32 %temp200, i32* %202
%new_rhythm201 = load i32*, i32** %new_rhythm
%203 = getelementptr i32, i32* %new_rhythm201, i32 43
%temp202 = load i32, i32* %203
%204 = getelementptr i32, i32* %116, i32 43
store i32 %temp202, i32* %204
%new_rhythm203 = load i32*, i32** %new_rhythm
%205 = getelementptr i32, i32* %new_rhythm203, i32 44
%temp204 = load i32, i32* %205
%206 = getelementptr i32, i32* %116, i32 44
store i32 %temp204, i32* %206
%new_rhythm205 = load i32*, i32** %new_rhythm
%207 = getelementptr i32, i32* %new_rhythm205, i32 45
%temp206 = load i32, i32* %207
%208 = getelementptr i32, i32* %116, i32 45
store i32 %temp206, i32* %208
%new_rhythm207 = load i32*, i32** %new_rhythm
%209 = getelementptr i32, i32* %new_rhythm207, i32 46
%temp208 = load i32, i32* %209
%210 = getelementptr i32, i32* %116, i32 46
```

```
store i32 %temp208, i32* %210
%new_rhythm209 = load i32*, i32** %new_rhythm
%211 = getelementptr i32, i32* %new_rhythm209, i32 47
%temp210 = load i32, i32* %211
%212 = getelementptr i32, i32* %116, i32 47
store i32 %temp210, i32* %212
%new_rhythm211 = load i32*, i32** %new_rhythm
%213 = getelementptr i32, i32* %new_rhythm211, i32 48
%temp212 = load i32, i32* %213
%214 = getelementptr i32, i32* %116, i32 48
store i32 %temp212, i32* %214
%new_rhythm213 = load i32*, i32** %new_rhythm
%215 = getelementptr i32, i32* %new_rhythm213, i32 49
%temp214 = load i32, i32* %215
%216 = getelementptr i32, i32* %116, i32 49
store i32 %temp214, i32* %216
%malloccall215 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32,
    i32* null, i32 1) to i32))
%217 = bitcast i8* %malloccall215 to i32*
%218 = getelementptr i32, i32* %217, i32 0
store i32 0, i32* %218
%219 = getelementptr i32, i32* %217, i32 0
%temp216 = load i32, i32* %219
%220 = getelementptr i32, i32* %116, i32 50
store i32 %temp216, i32* %220
store i32* %116, i32** %new_rhythm
%rhythm217 = load i32*, i32** %rhythm4
%i218 = load i32, i32* %i
%221 = getelementptr i32, i32* %rhythm217, i32 %i218
%222 = load i32, i32* %221
%new_rhythm219 = load i32*, i32** %new_rhythm
%i220 = load i32, i32* %i
%223 = getelementptr i32, i32* %new_rhythm219, i32 %i220
store i32 %222, i32* %223
%i221 = load i32, i32* %i
%tmp = add i32 %i221, 1
store i32 %tmp, i32* %i
```

```
  br label %while

merge:                                            ; preds = %while
  %instrument225 = load i32, i32* %instrument1
  %struct.ptr = getelementptr inbounds { i32*, i8**, i32, i32 }, { i32*, i8**,
      i32, i32 }* %res, i32 0, i32 3
  store i32 %instrument225, i32* %struct.ptr
  %size226 = load i32, i32* %size2
  %struct.ptr227 = getelementptr inbounds { i32*, i8**, i32, i32 }, { i32*,
      i8**, i32, i32 }* %res, i32 0, i32 2
  store i32 %size226, i32* %struct.ptr227
  %new_melody228 = load i8**, i8*** %new_melody
  %struct.ptr229 = getelementptr inbounds { i32*, i8**, i32, i32 }, { i32*,
      i8**, i32, i32 }* %res, i32 0, i32 1
  store i8** %new_melody228, i8*** %struct.ptr229
  %new_rhythm230 = load i32*, i32** %new_rhythm
  %struct.ptr231 = getelementptr inbounds { i32*, i8**, i32, i32 }, { i32*,
      i8**, i32, i32 }* %res, i32 0, i32 0
  store i32* %new_rhythm230, i32** %struct.ptr231
  %res232 = load { i32*, i8**, i32, i32 }, { i32*, i8**, i32, i32 }* %res
  ret { i32*, i8**, i32, i32 } %res232
}

define i32 @main() {
entry:
  %a = alloca { i32, i8* }
  %struct.ptr = getelementptr inbounds { i32, i8* }, { i32, i8* }* %a, i32 0,
      i32 1
  store i8* getelementptr inbounds ([4 x i8], [4 x i8]* @"4#.6", i32 0, i32 0),
      i8** %struct.ptr
  %struct.ptr1 = getelementptr inbounds { i32, i8* }, { i32, i8* }* %a, i32 0,
      i32 1
  %struct.val.test = load i8*, i8** %struct.ptr1
  %prints = call i32 (i8*, ...) @puts(i8* %struct.val.test)
  ret i32 0
}
```

```
declare noalias i8* @malloc(i32)
```

8.9 run.sh

```
#!/bin/bash
set -e
if [ -z "$1" ]
  then
    echo "Unrecognized usage "
    echo "For a list of sample usage: ./run.sh -h or ./run.sh -help"
    exit 1
fi
if [ "$1" = "-h" ]||[ "$1" = "-help" ]
    then
    echo "the caml run.sh accept three different usage"
    echo "./run.sh -s <input_file> or ./run.sh -song <input_file> "
    echo "convert the caml script into a midi file with <input_file>.midi"
    echo ""
    echo ""
    echo "./run.sh -p <input_file> or ./run.sh -print <input_file> "
    echo "print the output from compiling <input_file>"
    echo ""
    echo ""
    echo "./run.sh -o <input_file> or ./run.sh -txt <input_file> "
    echo "print the output from compiling into the text file <input_file>.txt"
    echo ""
    exit 1
fi
if [ "$1" = "-s" ]||[ "$1" = "-song" ]
    then
    f=$2
    n=${f%.cl*}
    cat $f | ./src/caml.native > "$n.ll"
    llc -relocation-model=pic "$n.ll"
    cc -o $n.exe "$n.s" "src/stdlib.o" "-lm"
    ./$n.exe > $n
```

```
    ./midifile/midi_generator -o ./$n
    rm $n
    rm $n.s
    rm $n.exe
    exit 0
fi
if [ "$1" = "-p" ]||[ "$1" = "-print" ]
    then
    f=$2
    n=${f%.cl*}
    cat $f | ./src/caml.native > "$n.ll"
    llc -relocation-model=pic "$n.ll"
    cc -o $n.exe "$n.s" "src/stdlib.o" "-lm"
    ./$n.exe
    rm $n.s
    rm $n.exe
    exit 0
fi
if [ "$1" = "-o" ]||[ "$1" = "-txt" ]
    then
    f=$2
    n=${f%.cl*}
    cat $f | ./src/caml.native > "$n.ll"
    llc -relocation-model=pic "$n.ll"
    cc -o $n.exe "$n.s" "src/stdlib.o" "-lm"
    ./$n.exe > ./$
    rm $n.s
    rm $n.exe
    exit 0
fi
```

8.10 testall.sh

```
#!/bin/bash

# Path to the LLVM interpreter
```

```
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="cc"

# Path to the CAML compiler
caml="src/caml.native"

TARGET="./target"

# Set time limit for all operations
ulimit -t 30

globallog=testall.log
rm -f $globallog
error=0
globalerror=0
passed=0
total=0
keep=0

Usage() {
    echo "Usage: testall.sh [options] [.cl files]"
    echo "-k   Keep intermediate files"
    echo "-h   Print this help"
    exit 1
}

SignalError() {
    if [ $error -eq 0 ] ; then
   error=1
    fi
}
```

```
# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile. Differences, if any, written to difffile
Compare() {
    generatedfiles="$generatedfiles $3"
    echo diff -b $1 $2 ">" $3 1>&2
    diff -b "$1" "$2" > "$3" 2>&1 || {
   SignalError "$1 differs"
   echo "FAILED $1 differs from $2" 1>&2
    }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
    echo $* 1>&2
    eval $* || {
   SignalError "$1 failed on $*"
   return 1
    }
}

# RunFail <args>
# Report the command, run it, and expect an error
RunFail() {
    echo $* 1>&2
    eval $* && {
   SignalError "failed: $* did not report an error"
   return 1
    }
    return 0
}

Check() {
    error=0
    basename=‘echo $1 | sed ’s/.*\\///
                          s/.cl//’‘
```

```
    reffile=‘echo $1 | sed ’s/.cl$//’‘
    basedir="‘echo $1 | sed ’s/\/[^\/]*$//’‘/."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles target/${basename}.s target/${basename}.exe
        target/${basename}.out" &&
    Run "$caml" "$1" ">" "target/${basename}.ll" &&
    Run "$LLC" "target/${basename}.ll" ">" "target/${basename}.s" &&
    Run "$CC" "-o" "target/${basename}.exe" "target/${basename}.s"
        "src/stdlib.o" "-lm" &&
    Run "target/${basename}.exe" > "target/${basename}.out" &&
    Compare target/${basename}.out ${reffile}.out target/${basename}.diff

    # Report the status and clean up the generated files

    #echo "before error check"
    total=$((total+1))
    if [ $error -eq 0 ] ; then
        if [ $keep -eq 0 ] ; then
            rm -f $generatedfiles
        fi
        echo -n "."
        echo "###### SUCCESS" 1>&2
        passed=$((passed+1))
    else
      printf "\n"
      echo -n "$basename..."
        echo "failed"
        echo "###### FAILED" 1>&2
        globalerror=$error
    fi
}
```

```
CheckFail() {
    error=0
    basename=‘echo $1 | sed ’s/.*\\///
                            s/.cl//’‘
    reffile=‘echo $1 | sed ’s/.cl$//’‘
    basedir="‘echo $1 | sed ’s/\/[^\/]*$//’‘/."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
    RunFail "$caml" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
    Compare ${basename}.err ${reffile}.err ${basename}.diff

    # Report the status and clean up the generated files
    total=$((total+1))
    if [ $error -eq 0 ] ; then
        if [ $keep -eq 0 ] ; then
            rm -f $generatedfiles
        fi
        echo -n "."
        echo "###### SUCCESS" 1>&2
        passed=$((passed+1))
    else
      printf "\n"
      echo -n "$basename..."
        echo "failed"
        echo "###### FAILED" 1>&2
        globalerror=$error
    fi
}

while getopts kdpsh c; do
    case $c in
   k) # Keep intermediate files
```

```
      keep=1
      ;;
   h) # Help
      Usage
      ;;
    esac
done

shift `expr $OPTIND - 1`

LLIFail() {
  echo "Could not find the LLVM interpreter \"$LLI\"."
  echo "Check your LLVM installation and/or modify the LLI variable in
     testall.sh"
  exit 1
}

which "$LLI" >> $globallog || LLIFail

if [ $# -ge 1 ]
then
    files=$@
else
    files="tests/test_*.cl"
fi

for file in $files
do
    case $file in
   *test_*)
      Check $file 2>> $globallog
      ;;
   *fail_*)
      CheckFail $file 2>> $globallog
      ;;
   *)
      echo "unknown file type $file"
```

```
      globalerror=1
      ;;
    esac
done
printf "\n"
echo $passed" out of "$total" tests passed!"

exit $globalerror
```