# BMWSA

Aman Chahar (ac3946)
Miao Yu (my2457)
Sikai Huang(sh3518)
Baokun Cheng(bc2651)

## Data Processing Language

# Table of Contents

7. Lessons Learned
8. Appendix

# 1. Introduction

Data Processing language is designed to handle complex file operations and data parsing implementations. We provide easy I/O operations for handling multiple files together with functions like merge, split, copy, write etc. that requires either high level language or very lengthy/cumbersome low level language syntax and complexities. Most file operations involve some of the data processing and thus, we provide flexibility in our language to do them with ease. As most of the reading and writing into files are done line by line, we have designed our language to handle these operations easily by dedicating data types like Line, Para etc. that specifically handle chunk of data together. For the scope of project we are focusing on text files, as these are one of the most common operations done by any useful program. Similar to most of the mainstream languages, we primarily support imperative programming. Syntax is made similar to C/C++ that will be compiled to LLVM code. Extension of the language would be .bmwsa and other details are explained in the following sections.

Some common functionalities:

1) Splitting of a file into 2 or more files

2) Merging of two files in one single file

3) Copying some lines from one text file to another file (by line number)

4) Deleting some lines a file (by line number)

5) Deleting/copying some lines based on query term

6) return size of a file(line), number of letters and size of a line(column)

Abstraction of files operations and optimized data processing will allow people without much computer science background to easily maintain the data they need. It can be extended to incorporate many optimizations making it suitable for both efficient file and data processing, computing and coding.

# 2. Language Tutorial

BMWSA (data processing language) is designed to address data processing challenges with minimal effort. We use C-like syntax for programmers to easily adapt to our language. Every program needs **main()** function as compiler treats that as starting point. Curly braces marks the starting and end of scope, parenthesis for function arguments and every statement ending with semicolon. Inline comments and multiple line comments are same as C syntax. Inline comments can be used by starting with // and multi-line comments are enclosed with /* and */. BMWSA allows the use of include statements with stdlib as its standard library.

```
/* Basic Hello World program */
// main function, which should exist in every program
int main()
{
        print("hello world");
        return 0;
}
```

BMWSA uses combination primitive and complex data types for representation. There are 4 primitive data types i.e. Int, Float, Boolean, Char and 3 complex data types i.e. String, File, Arrays. These variables can be global and local. They reside within their scope if local while global spans over all the program. However, global variables are overshadowed in the presence of local variables.

```
int a;
float b;
char c;
bool d
String e;
File f;
Int *arr;             /*array declaration*/
Arr = new[10]         // Array of 10 size
e = new [10]                  // Define an array whose length is 10 bytes
```

**Functions**

Functions are defined followed C-style. Functions are passed-by-value, which means the function will create a copy for each of its primitive argument and won't change the value after its execution. But for arrays

**Compiling and running the program**

Clone the github repository to get complete source code and pre-compiled executables.

Pre-requisites for running the code: -

**LLVM**
**OPAM**

Before executing the code run: -

```
eval `opam config env`
```

For pre-compiled executable use

```
./run_bmwsa <your_code>
```

To recompile the source code run make clean in the home directory: - bmwsa-llvm. Then make to get executable run_bmwsa file.

To output the LLVM IR of a given program, we need to run the binary on /../bmwsa/ directory and type “./bmwsa.native -l < the_program_you_want_to_compile > the_name_you_want_as_output“

To run the compiled code, we can simply type “lli name_of_output” to run it.
To test the test cases, we can type “./testall”  in that directory to see the result.

# 3. Language Manual

## 3.1 Lexical Elements

Token are separated using white space characters like space, tab or newline.

### 3.1.1 Identifiers

Identifiers are strings defined by sequence of letters and digits. Similar to C language, they should start with an alphabet and may contain underscore. Rules are defined as:

| | |
|---|---|
| |'0'-'9' | {digit} |
| |'a'-'z' | {lowercase_letter} |
| |'A'-'Z' | {uppercase_letter} |
| |lowercase_letter|uppercase_letter | {letter} |
| |(letter)(letter|digit|_)* | {identifier} |

### 3.1.2 Keywords

These keywords are reserved for use in the language and therefore cannot be used as identifiers. These keywords are case sensitive:

| | | | | | | |
|---|---|---|---|---|---|---|
| int | char | string | new | true | float | file |
| return | if | else | for | while | null | void |
| false | bool | include | | | | |

### 3.1.3 Literals

#### Character Literals

A character literal is expressed as a character enclosed in ASCII single quotes. However, there are few special 2 character literals that start with backslash ('\') that escapes the following character. Some of the special character literals are :-

'\n' - New Line

‘\t’, - Tab

‘\\’, backslash

\’,  escape single quote

\”, escape double quotes

‘\x2a’ escaped heximal characters

## String Literals

A string literal consists of zero or more characters enclosed in double quotes. They can be accessed using [] operator. Strings are defined as mutable internally and any change done by using [] will return new string.

**Example:** “abc”

String literals can also accept the escape characters like “\r\n”, or characters in hexadecimal code like “\x2a” or “\x2A”. In this way, the programmer can assign whatever binary code he wants for the string operations

## Integer Literals

An integer literal is expressed in decimal. It is defined as a consecutive set of digits in a string. Or in regular expression [‘0’ - ‘9’]+

**Example:** 2016

## Float Literals

Float literals are defined as an integer followed by a dot or a dot followed by integer, or the common float number we see in math.

Example: 20.  .2  20  .2

## Floating point Literals

Floating point literals are defined exactly like C.
“A floating constant consists of an integer part, a decimal point, a fraction part, an e, and an optionally signed integer exponent. The integer and fraction parts both consist of a sequence of digits. Either the integer part or the fraction part (not both) may be missing; either the decimal point or the e and the exponent (not both) may be missing. Every floating constant is taken to be double-precision.” C Language Reference manual

## 3.1.4 Comments

Comments are block of code enclosed between /* and */. You can define inline comments using // operator as described in next section. Comments can span across multiple lines.

## 3.1.5 Operators

| Symbol | Description |
|---|---|
| = | assign |
| +,-,*,/ | corresponding math operators |
| ++,-- | increment/decrement by one |
| % | modulus operator |
| ==,>,<,>=,<=,!= | comparison operators |
| “” | string identifier |
| ‘’ | character identifier |
| () | order enforcement operator |
| [] | index accessing |
| \|\| | Logic or |
| && | Logic and |
| ! | Logic not |
| // | Comment everything in that line following // |
| /* */ | Multiple lines comment |

# 3.2. Data Types

## 3.2.1 Primitive types

int: 32 bit signed integer. Range from -2,147,483,648 to 2,147,483,647. The first bit represents the sign, the others represent the value.

float: 64 bit double-precision floating-point number that follows IEEE 754 standard.

char: 8 bit integer (0-255), is used to represent the character. It can contain any 1-byte long ASCII characters

Bool: 1bit data type (i1 in LLVM), which is used for the branch conditions. It can either be true (1) or false (0). Logic operations can be implemented on boolean type

## 3.2.2 Arrays

Arrays are defined as a collection of primitive types. Arrays can be accessed directory by using [] operator. Their size can be increased, decreased as required.

**string:** a char array ends with \x0. It is implemented as a pointer to a char array in LLVM, but for the readability, we define it as a string

**file:** A file handle for file reading or writing. Similar to file type in C. It is implemented as a pointer

**Int array:** An array of integers. The size is defined by user. It is implemented as a pointer (i32 *) to integer type (i32) in LLVM.

**String array:** An array of string. Similar to int array, it is implemented as a pointer points to string, which is also a pointer to char (i8) in LLVM

The construction of arrays can be finished by writing Array_id=new [num], where the num refers to the number of the elements the user wants for the array and it must be an integer expression that is greater than zero. The initialization is realized through creating a type of pointer to the element in LLVM. The pointer is then then set to the available memory which is allocated on heap and the size of it is determined by the num and by the size of element type.

The reference to the elements in array can be realized by id[num]. The index num is the index of the element the user want. The num can be any expression that has an integer value.

# 3.3 Expressions and Statements

## 3.3.1 Expression

Expressions are statements that end with semicolon and usually comprises of assignments and function calls.

expression ;
**Example**
int a;
a=1;

## 3.3.2 if and if else statements

The if statements is used to conditionally execute part of program with true detected by system.

```
If expression
statement;
```

```
if  expression
{
        statement
}

// if else
if  expression
{ statement 1 }
else
{ statement 2 }
```

if statements detect the true value, if it meets it the statement 1 will be executed. However, it meets false value the statements 2 will be executed. The execution code without a braces.
Here is an actual example:

```
if x == 5
        { save ( output ," Output . txt “);}
```

if x == 5 is true, then the statement save ( output ," Output . txt "); will be executed. But if x == 5 is false another example is

```
if x == 5
        {save ( output ," Output . txt ");}
else
        {print ( id  * idf); }
```

above example shows that when if meets false the statements , in the else block, will be executed.

## 3.3.3 while statement

The while statement with an ending detection at the beginning of the block.
example:

```
//  while loop
while  expression
        {statement}
```

the while statement detect the expression first. If the expression is true, the statements will be executed. And the statements will be executed repeatedly till the system meets the state will never correspond to the expression.
The example below shows a while statement, which save out the file name with 1 to 5.

```
int file_name = 1
while (file_name <= 5)
        { /* Do some operations here); */
                Expressions
                i++
        }
```

## 3.3.4 for statement

The for statement is useful statement to iterate the program states.
**Example**:

```
//  for loop
for  (initialization:termination:increment)
{
        statement
}
```

The for statement evaluate the initialization first, then it evaluates the expression increment. Each increment loop, the for statement will detect the increment is correspond to the termination states. If is not meet the termination states, it will execute the statement. However, when it meet the state that correspond to the termination state, it will break out.

**Example:**

```
for ( i = 1 : i<= 5:i++){

        print(i);;
}
```

above example is the for- version of while loop. The system will print the integer 1 to 5 each in a line.

# 3.4 Functions

Custom functions can be defined using C type syntax

*return_type function_name( arguments)*

We also (want to) incorporate function overloading concept that allows same function name with different type of arguments. Ability to pass variable parameters can be done using by passing array.

Functions are called by using *function_name(arguments)*

# 3.5 Declarations

## 3.5.1 File operations

Our language requires handling of files and thus, requires extensive use of file handling operations. Some of the library functions we have are: -

**fopen(string FilePath, string Mode)**

Returns File pointer

By default open up in Read mode but other modes can be explicitly mentioned: -

“r”:open the file for reading

“r+”:update the file(read and write)

“w”: construct a new file for writing and clear the old file

“w+”: construct a new file for updating

“a”: append

**Example:**

file a = open(“abc.txt”);

file b = open(“newfile.txt”,”w”);

**API:**

**int close (File )**

Return 1 on success and 0 otherwise

Closes the File pointer and frees the space.

**int remove(string FilePath)**

Return 1 on success and 0 otherwise

Deletes the file associated with file path

**int rename(string FilePath, string NewName)**

Returns 1 on success and 0 otherwise

Renames the file associated with the file pointer

**void mergefile(string NewFile, string A, string B)**

Merges two files A and B to create a new file

**void copyfile(string initial_file ,string new_file)**

Copies the current file to new file

**int splitfile(string new_file_1 ,string new_file_2, string initial_file, int ln, int col)**

split the initial file into two separate files, new_file_1 and new_file_2, from the position with line number ln and column number col

**int size(string FilePath)**

Returns the size of the file, in byte.

**int feof(file FileStream)**

Tests the end-of-file indicator for the given stream.

Returns a non-zero value when end-of-file indicator associated with the stream is set, else zero is returned.

**int countLine(string FilePath)**

Returns total line number of the file

## 3.5.2 Formative output

**printx  (const char *format, ...)**

BMWSA's printx has same syntax as C's printf

## 3.5.3 Character/String I/O function

**char fgetc(file FileStream)**

Returns the next character from the specified stream and advances the position indicator for the stream.

**int getLine(string file, int ln)**

Print the content of the line with line number ln.

Returns 0 if the ln is out of boundary, 1 otherwise.

**void getChar(string FilePath, int ln, int col)**

Print the character at the specific position with line number ln and column number col.

**char fputc(char c, file FileStream)**

Writes a character specified by the argument char to the specified stream and advances the position indicator for the stream.

If there are no errors, the same character that has been written is returned. If an error occurs, EOF is returned and the error indicator is set.

**int fputs(string s, file FileStream)**

Writes a string to the specified stream.

**void insert(string FiliePath, string content, int ln, int col)**

Insert content into the position(ln, col) of the file.

**void deleteLine(string FiliePath, int start, int end)**

Delete lines from file from start to end.

## 3.5.4 String functions

**int lengthOf(string str)**

Returns the length of the string str.

**string *splitstring(string a,int c,string b)**

Return an array of string that are separated by the tokens in string b. The size of the array is determined by the integer c, and the string a is the string for split.

For example

splitstring(“Miao Baokun Aman Sikai”,3,” “)

Will give an array of 3 strings, “Miao”, “Baokun” and “Aman”.

**string atoi(string str)**

Coverts the string argument str into an integer and returns it.

**string itos(int t)**

Coverts the int argument t into a string and returns it.

**bool isCharPresent(string str, char a)**

Returns true if character a is in string str, otherwise returns false.

**int replacewords(string FilePath, string ini_word, string new_word)**

Replace the ini_word in file with new_word. All these words must contain only alphanumeric characters.

Returns count of ini_word.

**int searchwords(string FilePath, string word)**

Scan the whole file and returns the count of word. Word must contain only alphanumeric characters.

**int deleteword(string FilePath, string word)**

Scan the whole file and delete the word in argument. Word must contain only alphanumeric characters.

Returns the count of word.

**void searchPattern(file File, string stopsign)**

Scan the file from the File pointer, and stops until meet the first stopsign

**string extractBefore(file File, file obj, string stopsign, string del)**

Scan the first file from the File pointer, until meet the first stopsign. Store the string before the stopsign excluding characters in the fourth argument del. Return this string.

**string putStopChar(file File, file obj, string stopsign, string del)**

Scan the first file from the File pointer, until meet any character of the stopsign. Store the string before the stopsign excluding characters in the fourth argument del. Return this string.

# 4. **Project Plan**

## 4.1 Process

Our team met once a week with TA Julie to discuss project logistics, deadlines, milestones and future steps. These meetings helped us maintaining steady progress on the project. We faced problems in deciding goal of our language as we could either focus on string operations or file operations. We decided to go with file operations as many existing languages like R, Python focus on handling string operations. We all worked together initially while writing proposal and deciding the Language Reference Manual. We coordinated initial work through Google docs, email and wechat to work for Language Reference Manual and Proposal. Once coding and testing started we shifted to Github to manage our project.

LLVM was new to us so we invested lot of time to understand the moe library and generating LLVM through it. We divided functionalities of our language among ourselves so that everyone is accountable for one function from design to testing. This ensured that everyone was working parallely.

### 4.2 Testing

Every group member was responsible for creating test cases for their functionality and testing before pushing the code to the repository. Full testing of our compiler was possible only last few days when we got all the components combined. All these test cases comprised both pass and fail cases.

### 4.3 Programming Style Guide

As we all were working on different modules simultaneously we had to use some programming conventions to maintain readability and consistency in the code

1. Use spaces for indentation instead of tabs
2. Use comments for every new feature
3. Grouping of code to separate different sections
4. Using camelCase for defining functions

5. Lowercase for variables

## 4.4 Timeline

| | |
|---|---|
| Feb 1 | Finalizing of Team |
| Feb 10 | Submission of Project Proposal |
| Feb 27 | Setting up of repository |
| Feb 29 | Adding Project ideas for LRM |
| Mar 7 | Language Reference Manual |
| Mar 22 | Completion of basic scanner and Parser |
| April 6 | Hello World Program milestone |
| April 15 | Automated testing script |
| April 22 | Data types added |
| April 26 | Printf, file open and other basic I/O functions |
| May 6 | Library Functions and Arrays |
| May 9 | Testing and Semantic check |
| May 11 | Project Presentation and Final project submission |

## 4.5 Roles and Responsibilities

Initially everyone worked on project proposal and Language reference manual. As the code generation started we divided various functionalities among different members which they tested on their own. Most of the times coding was done while sitting together where everyone helped each other and exchanged roles depending upon availability and completion of their tasks.

Aman Chahar

- Project Manager

- Project Proposal
- LRM
- Code Generation
- Test suite

Miao Yu

- Project proposal
- LRM
- Code Generation
- Parser
- Scanner
- Test suite

Sikai Huang

- Project proposal
- LRM
- Test suite

Baokun Cheng

- Project Proposal
- LRM
- Library design
- Test suite

Weiduo Sun

- Project proposal
- LRM
- Final Report

## 4.6 Software development environment

- Ubuntu 15: - Everyone used same distribution to maintain consistency
- Ocaml: - To code our complete compiler. Packages like LLVM to generate final code
- LLVM: Version 3.6 to compile our generated code
- Bash: - Writing scripts to automate test cases and run our compiled code
- Google docs/drive: - To work on Final report, Proposal  and LRM
- Sublime Text and vim for writing code00

## 4.7 Git Commit Log

```
commit 7efd28fc83a5420267ae78468f18e051f5661de0
Author: Aman Chahar <amaniiitd@gmail.com>
Date:   Wed May 11 05:03:56 2016 -0400

    Added Runnable script

commit 9e48f102713cfc4a43b703337fe29639d49af2f4
Author: my2457 <my2457@columbia.edu>
Date:   Wed May 11 00:45:00 2016 -0400

    strtok->split

commit e29696d57737d215d851e76beaed4935512a8cb5
Author: CBKzz <baokuncheng@gmail.com>
Date:   Tue May 10 20:23:48 2016 -0700

    Add files via upload

commit 78e5e4eea0c1ab5f2bafde97260ce51b988205a3
Author: my2457 <my2457@columbia.edu>
Date:   Tue May 10 16:19:11 2016 -0400

    Arrays + split

commit 51889369e4c91672160097eff5dbff36818491fd
Author: my2457 <my2457@columbia.edu>
Date:   Tue May 10 15:24:50 2016 -0400

    Char operations

commit 363c12a5a38200ee84bf93ede50530ed7eec4760
Author: my2457 <my2457@columbia.edu>
Date:   Tue May 10 15:08:47 2016 -0400

    Arrays

commit b904eb7e8e99ed3e38be7ac6c39f2aca8c5ff00a
Merge: 3f18c72 8080e18
Author: Aman Chahar <amaniiitd@gmail.com>
Date:   Tue May 10 14:35:09 2016 -0400

    Merge branch 'master' of https://github.com/amanchahar/PLT-Data-Processing-Langauge

commit 3f18c7287e1125ee229641a2d435f3d0be1cc534
Author: Aman Chahar <amaniiitd@gmail.com>
Date:   Tue May 10 14:34:56 2016 -0400
```

```
    Latest changes

commit 8080e182a727b3ebf5bf6fdd0d5d1980549aebcb
Author: CBKzz <baokuncheng@gmail.com>
Date:   Mon May 9 19:27:04 2016 -0700

    insert file, search word, replace, delete

commit 9d1d593902e7fd0444fd2a62cacb34f43db187d5
Author: Aman Chahar <amaniiitd@gmail.com>
Date:   Mon May 9 19:20:05 2016 -0400

    Merged Array Part

commit 1cfbc05257da8354621e25617d2cbfa2259f264b
Author: Aman Chahar <amaniiitd@gmail.com>
Date:   Mon May 9 17:53:10 2016 -0400

    Include library

commit 082e0b8947a1a684049d635ef1020c8784cfd2d6
Author: my2457 <my2457@columbia.edu>
Date:   Mon May 9 15:30:29 2016 -0400

    string/char

commit da88ee67b588e17a3999f4b1f03460b1d09b32c2
Author: CBKzz <baokuncheng@gmail.com>
Date:   Mon May 9 10:58:42 2016 -0700

    modified return type

commit ef2ec492fd86ca77bef1a615937e4332bbde5b73
Author: CBKzz <baokuncheng@gmail.com>
Date:   Mon May 9 08:33:06 2016 -0700

    remove, rename

commit fa7ff99fe713b6553e50901c9cd907df5dca2234
Author: CBKzz <baokuncheng@gmail.com>
Date:   Mon May 9 08:32:49 2016 -0700

    deleteLines

commit 438e54c7efb7bd3c55c40796b439a39f9d125709
Author: my2457 <my2457@columbia.edu>
Date:   Mon May 9 07:48:28 2016 -0400

    array + memcpy
```

```
commit 3f3fe152f83bfc0b7f15bc11ab91c92b3ed16494
Author: CBKzz <baokuncheng@gmail.com>
Date:   Sun May 8 21:55:07 2016 -0700

    split file

commit 4f1a4700a7387e7bd358732076ecf6de336a3e20
Author: CBKzz <baokuncheng@gmail.com>
Date:   Sun May 8 20:42:51 2016 -0700

    merge file, print line

commit 7946b1e242a4ac6e8635c5e0a544067547e721ba
Author: CBKzz <baokuncheng@gmail.com>
Date:   Sun May 8 15:54:09 2016 -0700

    stdlib functions

commit 7002674e22ecc248ebdffd662320586e40fa9e55
Author: Aman Chahar <amaniiitd@gmail.com>
Date:   Sun May 8 17:45:51 2016 -0400

    Merged codes

commit a6be4afc576711868f8a1b946e798e86c08bf67a
Author: my2457 <my2457@columbia.edu>
Date:   Sun May 8 12:04:58 2016 -0400

    ++id --id

commit 82c63939dfe9b8c17d4db6138b625421cd77912b
Author: CBKzz <baokuncheng@gmail.com>
Date:   Sun May 8 09:00:03 2016 -0700

    file type

commit 3ad49c3b1ce6292454b693ec879bfde3a17631a3
Author: CBKzz <baokuncheng@gmail.com>
Date:   Sun May 8 08:57:09 2016 -0700

    read character

commit 16bf3310879d4c7d131c4c116a7c697e0870a258
Author: my2457 <my2457@columbia.edu>
Date:   Sun May 8 11:34:59 2016 -0400

    bypass all sement

commit 56fd9e1e1a29c4134215604accb17563a9817028
```

```
Author: my2457 <my2457@columbia.edu>
Date:   Sun May 8 11:26:55 2016 -0400

    size function

commit 4e5d583906127ded416ba453f26bfe23fae9882c
Author: my2457 <my2457@columbia.edu>
Date:   Sun May 8 10:47:42 2016 -0400

    new lib function & sement bypass

commit b5470d57de4a4899d3f52a4270fb005056891d0b
Author: my2457 <my2457@columbia.edu>
Date:   Sat May 7 16:54:05 2016 -0400

    fopen/fputs/math on double/

commit 73b3e6869f5ae0c3ebe0920958ac75cff2f16881
Merge: a1f19e6 ef69370
Author: Aman Chahar <amaniiitd@gmail.com>
Date:   Sat May 7 02:39:44 2016 -0400

    Float datatype Merge branch 'master' of
https://github.com/amanchahar/PLT-Data-Processing-Langauge

commit a1f19e66d12ef1b9ac0c28bc93649582079e7a37
Author: Aman Chahar <amaniiitd@gmail.com>
Date:   Sat May 7 02:39:09 2016 -0400

    Adding float datatype

commit ef6937058ff3fa363510bf2d82029ae17671371f
Author: my2457 <my2457@columbia.edu>
Date:   Fri May 6 21:07:56 2016 -0400

    Added files via upload

commit d38e736372565927b498b80fe2c48e3a9e3fcd20
Author: my2457 <my2457@columbia.edu>
Date:   Fri Apr 15 14:26:56 2016 -0400

    Added files via upload

commit d573b9c4e8c589a2e02482d3bfd6f4cf412c78d2
Author: my2457 <my2457@columbia.edu>
Date:   Fri Apr 15 14:25:01 2016 -0400

    Added files via upload

    comment tests
```

```
commit 11e9c82e381ff210a7415e7c1195aaeda22bc2f1
Author: Aman Chahar <amaniiitd@gmail.com>
Date:   Wed Apr 6 23:49:23 2016 -0400

    BMWSA extension files

commit c7142feb2871f00346d7f66a1be6490e9c357e57
Author: Aman Chahar <amaniiitd@gmail.com>
Date:   Wed Apr 6 23:36:17 2016 -0400

    Delete all temp files

commit f72a86bebe1cb1e6872680c94aaa5aa03ffece4c
Author: amanchahar <amaniiitd@gmail.com>
Date:   Wed Apr 6 23:34:23 2016 -0400

    removing .bash_logout

commit e908033bc2d8f229cb8893798662acfeb700d263
Author: amanchahar <amaniiitd@gmail.com>
Date:   Wed Apr 6 23:33:11 2016 -0400

    Deleting . files

commit a13ad2ed5911b5c8943721b7bd9381ba699eef37
Author: amanchahar <amaniiitd@gmail.com>
Date:   Wed Apr 6 23:28:59 2016 -0400

    Removing microc

commit d5b0f57ffe07b77937eb5eaccb1c50f0dfa10a51
Merge: 4dd3534 d985e05
Author: amanchahar <amaniiitd@gmail.com>
Date:   Wed Apr 6 23:25:58 2016 -0400

    Merge https://github.com/amanchahar/PLT-Data-Processing-Langauge

    Changing MicroC to BMWSA

commit 4dd3534c4f9d39245f9372b13ed3b99dd117527d
Author: amanchahar <amaniiitd@gmail.com>
Date:   Wed Apr 6 23:25:08 2016 -0400

    Changed to BMWSA

commit d985e0527b1bd58ea7cd70c322948d34dc0114ef
Author: Aman Chahar <Chahar@Amans-MacBook-Pro.local>
Date:   Wed Apr 6 13:35:38 2016 -0400
```

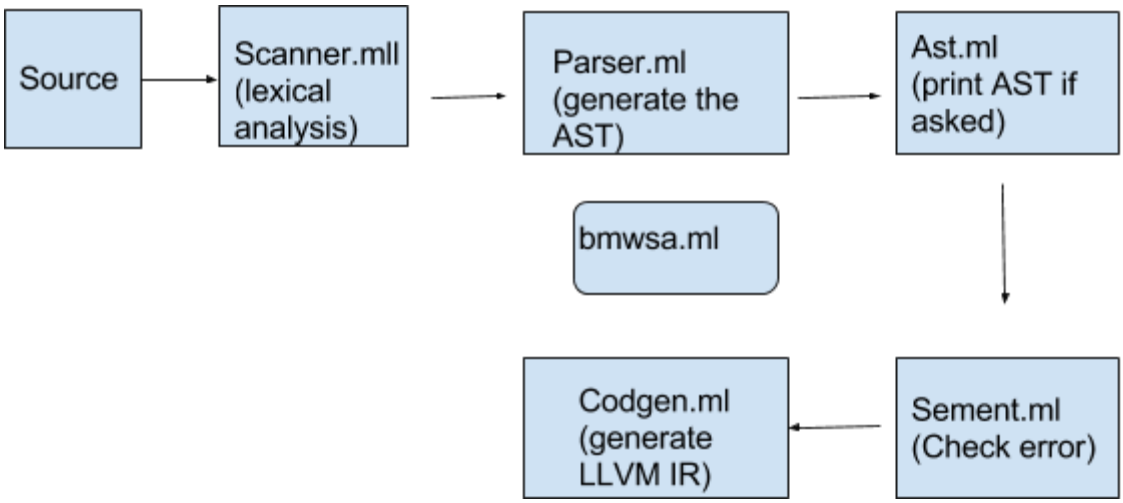
```
    Hello World

commit ac5eec5c82daf70462fcf239e9c31b0231b40413
Author: amanchahar <amaniiitd@gmail.com>
Date:   Wed Apr 6 12:30:37 2016 -0400

    Hello World

commit 5f96c7936dc69be7cd8300eb8090a755490720ff
Author: amanchahar <amaniiitd@gmail.com>
Date:   Wed Apr 6 12:25:33 2016 -0400

    Hello World

commit 4e213b49f6f3b2f28f9f457f5254c6d2a3872e8b
Author: Aman Chahar <Chahar@dyn-160-39-210-40.dyn.columbia.edu>
Date:   Tue Mar 8 01:50:55 2016 -0500

    Language Reference Manual

commit acd5475d2d4c82b6f078b742e1304d991a84ab3a
Author: Aman Chahar <amaniiitd@gmail.com>
Date:   Sat Feb 27 00:28:42 2016 -0500

    Updated language ideas

commit 46b3d6f131de9b2bad304d32121e7b4d83d3e30c
Author: Aman Chahar <amaniiitd@gmail.com>
Date:   Sat Feb 27 00:19:41 2016 -0500

    Renamed Proposal File

commit ee996a1af5eefb6f7e1814264d2d15bf7f222af0
Author: Aman Chahar <amaniiitd@gmail.com>
Date:   Sat Feb 27 00:16:38 2016 -0500

    Project Proposal

commit 9f0021ccf18d84af466a7149464b672042cdae5b
Author: amaniiitd <amaniiitd@gmail.com>
Date:   Sat Feb 27 00:10:19 2016 -0500

    Initial commit
```

# 5. Architecture

## 5.1 Compiler architecture

The code in compiler is mainly divided into 5 separate parts: scanner.mll, parser.mly, ast.ml, sement.ml and codegen.ml.

In scanner, we defined the tokens that we care and Ocaml will do the lexical analysis on the language the programmer writes and convert them to a set of tokens. Because the types of tokens are defined in parser, the parser will then build the abstract syntax tree on those tokens according to the grammars we defined inside. In ast.ml, we defined the methods to print them so that we can view the tree if necessary. And the sement.ml will check on the abstract syntax tree to see if there's any inconsistency with the program we wrote, and if there is, it will raise exceptions and tell the user what sepcific errors are and terminate the compiling. If there are no errors found by sement.ml, the AST will be passed further to codegen.ml, which will look into the patterns in AST and call the corresponding APIs of LLVM Ocaml binding to generate the expected code in LLVM. The overall flowchart of the program is defined in bmwsa.ml, which can also be illustrated on the following chart:




## 5.2 Scanner

We used regular expression to define the keywords of our program and defined the tokens for the lexical analysis. When Ocaml scans a matched pattern, it goes to parser to look for further explanation.

## 5.3 Parser

We defined the grammar of our language in parser.mly and the types, such as the syntax define/call a function, the syntax to initialize an array

## 5.4 Ast

In ast.ml, we defined the method to print the Abstract syntax tree of our program. Also, it contains the patterns which are used in codegen to do the pattern matching to call the corresponding LLVM-Ocaml APIs to generate the code

## 5.5 Semant

The sement.ml will check the if the program's syntax and types are following the principle of the language, and once it finds the discrepencies, it will stop and tell the user where the problem is.

## 5.6 bmwsa

It integrates the code in the aforementioned files and call the functions in them to finish the task of compiler. The compiler will compile the *.bmwsa file into LLVM IR, which can be run by lli or be compiled to binary by llc

# 6. Testing

We started with Micro C testing cases and modified the outputs according to our values. We divided our test suites into success and fail test cases. Success test cases were used to test if the current output was produced and program was behaving as it should be. Fail test cases were introduced to detect any syntactic errors which shouldn't be allowed giving garbage values. All the test cases were put in tests/ folder with prefix test and fail. Corresponding output files were also produced with same prefix. Automated script was used to evaluate all these test cases together.

Testing script: -

```
#!/bin/sh

LLI="lli"

BMWSA="./bmwsa.native"


ulimit -t 30

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

keep=0

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

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

```
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() {
    echo $* 1>&2
    eval $* || {
       SignalError "$1 failed on $*"
       return 1
    }
}


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/.bmwsa//'`
    reffile=`echo $1 | sed 's/.bmwsa$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

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

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.out" &&
    Run "$BMWSA" "<" $1 ">" "${basename}.ll" &&
    Run "$LLI" "${basename}.ll" ">" "${basename}.out" &&
    Compare ${basename}.out ${reffile}.out ${basename}.diff

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
```

```
        if [ $keep -eq 0 ] ; then
            rm -f $generatedfiles
        fi
        echo "OK"
        echo "###### SUCCESS" 1>&2
    else
        echo "###### FAILED" 1>&2
        globalerror=$error
    fi
}

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

    echo -n "$basename..."

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

    generatedfiles=""

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

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
        if [ $keep -eq 0 ] ; then
            rm -f $generatedfiles
        fi
        echo "OK"
        echo "###### SUCCESS" 1>&2
    else
        echo "###### FAILED" 1>&2
        globalerror=$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-*.bmwsa tests/fail-*.bmwsa"
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

exit $globalerror
```

## Demo Test case files

We used two files worldcup.txt and 2013films.txt file for final demo and testing. These files are wikipedia pages used for evaluating data processing tasks. This was part of CSDS course

offered at Columbia University where they asked to use AWK GREP SED and Data Wrangler used by Stanford.

## WorldCup.txt file: - Contains soccer statistics of all countries

```
! Team !! Titles !! Runners-up !! Third place !! Fourth place !! Top
4 <br/> finishes
|-
|bgcolor=#FFF68F|{{fb|BRA}}
|bgcolor=#FFF68F|5 ([[1958 FIFA World Cup|1958]], [[1962 FIFA World
Cup|1962]], [[1970 FIFA World Cup|1970]], [[1994 FIFA World
Cup|1994]], [[2002 FIFA World Cup|2002]])
|2 ([[1950 FIFA World Cup|1950]][[#1|*]], [[1998 FIFA World
Cup|1998]])
|2 ([[1938 FIFA World Cup|1938]], [[1978 FIFA World Cup|1978]])
|1 ([[1974 FIFA World Cup|1974]])
|10
|-
|bgcolor=#FFF68F|{{fb|ITA}}
|bgcolor=#FFF68F|4 ([[1934 FIFA World Cup|1934]][[#1|*]], [[1938 FIFA
World Cup|1938]], [[1982 FIFA World Cup|1982]], [[2006 FIFA World
Cup|2006]])
|2 ([[1970 FIFA World Cup|1970]], [[1994 FIFA World Cup|1994]])
|1 ([[1990 FIFA World Cup|1990]][[#1|*]])
|1 ([[1978 FIFA World Cup|1978]])
|8
|-
|bgcolor=#FFF68F|{{fb|GER}}[[#2|^]]
|bgcolor=#FFF68F|3 ([[1954 FIFA World Cup|1954]], [[1974 FIFA World
Cup|1974]][[#1|*]], [[1990 FIFA World Cup|1990]])
|4 ([[1966 FIFA World Cup|1966]], [[1982 FIFA World Cup|1982]],
[[1986 FIFA World Cup|1986]], [[2002 FIFA World Cup|2002]])
|4 ([[1934 FIFA World Cup|1934]], [[1970 FIFA World Cup|1970]],
[[2006 FIFA World Cup|2006*]], [[2010 FIFA World Cup|2010]])
|1 ([[1958 FIFA World Cup|1958]])
|12
|-
|bgcolor=#FFF68F|{{fb|ARG}}
|bgcolor=#FFF68F|2 ([[1978 FIFA World Cup|1978]][[#1|*]], [[1986 FIFA
World Cup|1986]])
|2 ([[1930 FIFA World Cup|1930]], [[1990 FIFA World Cup|1990]])
```

```
|align=center| —
|align=center| —
|4
|-
|bgcolor=#FFF68F|{{fb|URU}}
|bgcolor=#FFF68F|2 ([[1930 FIFA World Cup|1930]][[#1|*]], [[1950 FIFA
World Cup|1950]])
|align=center| —
|align=center| —
|3 ([[1954 FIFA World Cup|1954]], [[1970 FIFA World Cup|1970]],
[[2010 FIFA World Cup|2010]])
|5
|-
|bgcolor=#FFF68F|{{fb|FRA}}
|bgcolor=#FFF68F|1 ([[1998 FIFA World Cup|1998]][[#1|*]])
|1 ([[2006 FIFA World Cup|2006]])
|2 ([[1958 FIFA World Cup|1958]], [[1986 FIFA World Cup|1986]])
|1 ([[1982 FIFA World Cup|1982]])
|5
|-
|bgcolor=#FFF68F|{{fb|ENG}}
|bgcolor=#FFF68F|1 ([[1966 FIFA World Cup|1966]][[#1|*]])
|align=center| —
|align=center| —
|1 ([[1990 FIFA World Cup|1990]])
|2
|-
|bgcolor=#FFF68F|{{fb|ESP}}
|bgcolor=#FFF68F|1 ([[2010 FIFA World Cup|2010]])
|align=center| —
|align=center| —
|1 ([[1950 FIFA World Cup|1950]])
|2
|-
|{{fb|NED}}
|align=center| —
|3 ([[1974 FIFA World Cup|1974]], [[1978 FIFA World Cup|1978]],
[[2010 FIFA World Cup|2010]])
|align=center| —
|1 ([[1998 FIFA World Cup|1998]])
|4
|-
|{{fb|TCH}}<sup>[[#3|#]]</sup>
```

```
|align=center| —
|2 ([[1934 FIFA World Cup|1934]], [[1962 FIFA World Cup|1962]])
|align=center| —
|align=center| —
|2
|-
|{{fb|HUN}}
|align=center| —
|2 ([[1938 FIFA World Cup|1938]], [[1954 FIFA World Cup|1954]])
|align=center| —
|align=center| —
|2
|-
|{{fb|SWE}}
|align=center| —
|1 ([[1958 FIFA World Cup|1958]][[#1|*]])
|2 ([[1950 FIFA World Cup|1950]], [[1994 FIFA World Cup|1994]])
|1 ([[1938 FIFA World Cup|1938]])
|4
|-
|{{fb|POL}}
|align=center| —
|align=center| —
|2 ([[1974 FIFA World Cup|1974]], [[1982 FIFA World Cup|1982]])
|align=center| —
|2
|-
|{{fb|AUT}}
|align=center| —
|align=center| —
|1 ([[1954 FIFA World Cup|1954]])
|1 ([[1934 FIFA World Cup|1934]])
|2
|-
|{{fb|POR}}
|align=center| —
|align=center| —
|1 ([[1966 FIFA World Cup|1966]])
|1 ([[2006 FIFA World Cup|2006]])
|2
|-
|{{fb|USA}}
|align=center| —
```

```
|align=center| —
|1 ([[1930 FIFA World Cup|1930]])
|align=center| —
|1
|-
|{{fb|CHI}}
|align=center| —
|align=center| —
|1 ([[1962 FIFA World Cup|1962]][[#1|*]])
|align=center| —
|1
|-
|{{fb|CRO}}
|align=center| —
|align=center| —
|1 ([[1998 FIFA World Cup|1998]])
|align=center| —
|1
|-
|{{fb|TUR}}
|align=center| —
|align=center| —
|1 ([[2002 FIFA World Cup|2002]])
|align=center| —
|1
|-
|{{fb|YUG}}<sup>[[#3|#]]</sup>
|align=center| —
|align=center| —
|align=center| —
|2 ([[1930 FIFA World Cup|1930]], [[1962 FIFA World Cup|1962]])
|2
|-
|{{fb|URS}}<sup>[[#3|#]]</sup>
|align=center| —
|align=center| —
|align=center| —
|1 ([[1966 FIFA World Cup|1966]])
|1
|-
|{{fb|BEL}}
|align=center| —
|align=center| —
```

```
|align=center| —
|1 ([[1986 FIFA World Cup|1986]])
|1
|-
|{{fb|BUL}}
|align=center| —
|align=center| —
|align=center| —
|1 ([[1994 FIFA World Cup|1994]])
|1
|-
|{{fb|KOR}}
|align=center| —
|align=center| —
|align=center| —
|1 ([[2002 FIFA World Cup|2002]][[#1|*]])
|1
|-
:<div id="1">''<nowiki>*</nowiki> = hosts''
:<div id="2">''<nowiki>^</nowiki> = includes results [[List of men's
national association football
teams#Former_national_football_teams|representing West Germany
between 1954 and 1990]]''</div>
:<div id="3">''<sup><nowiki>#</nowiki></sup> = [[List of men's
national association football
teams#Former_national_football_teams|states that have since split
into two or more independent nations]]''<ref name="successor"/></div>
```

## Our implementation of the code: -

```
include("stdlib");



void worldcup(string path){
	file f;
	file temp;
	file result;
	string name;
	string check;
	int t;
	int rank;
```

```
        int i;char c;string year;bool flag;

        f=fopen(path,"r");
        temp=fopen("HopeNoFileGetsThisName.txt","w");
        result=fopen("worldcupresult.txt","w");

        while(!feof(f)){

                c=fgetc(f);
                searchPattern(f,"{{fb|");
                name=extractBefore(f,temp,"}}","");
                searchPattern(f,"#FFF68F|");
                rank=0;
                t=atoi(extractBefore(f,temp," ",""));
                c='(';
                while(c=='('){
                                i=0;
                                if(rank!=0) {
                                                searchPattern(f,"|");
                                                check=putStopChar(f,temp," \n","");
                                                t=atoi(check);
                                }
                        if(strcmp(check,"*]])")!=0) c=fgetc(f);
                        if(c=='('){
                                while(i<t){
                                        searchPattern(f,"FIFA World Cup|");
                                        year=extractBefore(f,temp,"]","");
                                        i++;
                                        fputs(name,result);
                                        fputs(",   ",result);
                                        fputs(itos(rank+1),result);
                                        fputs(",   ",result);
                                        fputs(year,result);
                                        fputs("\n",result);
                                }
                                if(strcmp(check,"*]])")!=0) rank++;
                        }

                        if( strcmp(check,"align=center|")==0 || strcmp(check,"*]])")==0){c='(';}
                        if(strcmp(check,"align=center|")==0)  {rank++;}
                }
        }
}

int main()
{
        worldcup("worldcup.txt");

        return 0;
}
```

Output file generated by the code: -

```
BRA,   1,   1958
BRA,   1,   1962
BRA,   1,   1970
BRA,   1,   1994
BRA,   1,   2002
BRA,   2,   1950
BRA,   2,   1998
BRA,   3,   1938
BRA,   3,   1978
BRA,   4,   1974
ITA,   1,   1934
ITA,   1,   1938
ITA,   1,   1982
ITA,   1,   2006
ITA,   2,   1970
ITA,   2,   1994
ITA,   3,   1990
ITA,   4,   1978
GER,   1,   1954
GER,   1,   1974
GER,   1,   1990
GER,   2,   1966
GER,   2,   1982
GER,   2,   1986
GER,   2,   2002
GER,   3,   1934
GER,   3,   1970
GER,   3,   2006*
GER,   3,   2010
GER,   4,   1958
ARG,   1,   1978
ARG,   1,   1986
ARG,   2,   1930
ARG,   2,   1990
URU,   1,   1930
URU,   1,   1950
URU,   4,   1954
```

```
URU,    4,    1970
URU,    4,    2010
FRA,    1,    1998
FRA,    2,    2006
FRA,    3,    1958
FRA,    3,    1986
FRA,    4,    1982
ENG,    1,    1966
ENG,    4,    1990
ESP,    1,    2010
ESP,    4,    1950
```

## 2013films.txt file: - It contains all the films and their details released in 2013. This is wikipedia page which is unformatted text file. We extract name, publisher and director from each movie.

### Input file: - (a small snapshot of the actual input file)

```
==2013 films==

===January – March===
<!-- DO NOT ADD FILMS WITHOUT A RELIABLE SOURCE -->
{| class="wikitable"
|- style="background:#b0e0e6; text-align:center;"
! colspan="2" | Opening
! style="width:17%;"| Title
! style="width:16%;"| Studio
! Cast and crew
! style="width:10%;"| Genre
! Ref.
|-
! rowspan="20" style="text-align:center; background:#ffa07a; textcolor:#000;"|J<br />A<br
/>N<br />U<br />A<br />R<br />Y
| rowspan="3" style="text-align:center; background:#ffdacc; textcolor:#000"| 4
| ''[[A Dark Truth]]'' || [[Magnolia Pictures]] || [[Damian Lee]] (director/screenwriter);
[[Andy Garcia]], [[Kim Coates]], [[Eva Longoria]], [[Forest Whitaker]], [[Deborah Kara
Unger]], [[Devon Bostick]], [[Al Sapienza]], [[Steven Bauer]], [[Kevin Durand]], [[Claudette
Lali]] || [[Action film|Action]] || <center><ref>{{cite
web|url=http://www.comingsoon.net/films.php?id=95169 | title=A Dark Truth |
publisher=ComingSoon.net | accessdate=2013-01-03}}</ref></center>
|-
| ''[[Table No. 21]]'' || [[Eros International]] || Aditya Datt (director); Jimmy-sen
(screenplay); [[Tena Desae]], [[Rajeev Khandelwal]], [[Paresh Rawal]] || [[Thriller
film|Thriller]] || <center><ref>{{cite web |
url=http://boxofficemojo.com/movies/?id=tableno21.htm | title=Table No. 21 |
publisher=BoxOfficeMojo.com | accessdate=2013-01-8}}</ref></center>
|-
```

```
| ''[[Texas Chainsaw 3D]]'' || [[Lions Gate Entertainment|Lionsgate]] || [[John Luessenhop]]
(director); Kirsten Elms, [[Adam Marcus (director)|Adam Marcus]], Debra Sullivan (screenplay);
[[Alexandra Daddario]], Dan Yeager, [[Trey Songz]], [[Scott Eastwood]], [[Tania Raymonde]] ||
[[Horror film|Horror]] || <center><ref>{{cite web |
url=http://www.comingsoon.net/films.php?id=77377 | title=Texas Chainsaw 3D |
publisher=ComingSoon.net | accessdate=2012-04-12}}</ref></center>
|-
| style="text-align:center; background:#ffdacc; textcolor:#000;"| 8
| ''[[The Grandmaster (film)|The Grandmaster]]'' || Block 2 Pictures / Jet Tone Films ||
[[Wong Kar-wai]] (director/screenplay); [[Tony Leung Chiu-Wai]], [[Zhang Ziyi]], [[Song
Hye-kyo]], [[Chang Chen]], [[Zhao Benshan]], [[Yuen Woo-ping]], [[Xiaoshenyang]], [[Cung Le]],
[[Lo Hoi-pang]], [[Leung Siu-lung]], [[Julian Cheung]], [[Lo Mang]], [[Berg Ng]] || [[Action
film|Action]], [[Drama film|Drama]] || <center><ref>{{cite web |
url=http://www.berlinale.de/en/presse/pressemitteilungen/alle/Alle-Detail_16084.html |
title=The Grandmaster | publisher=Berlinale.de | accessdate=2013-02-22}}</ref></center>
|-
| rowspan="4" style="text-align:center; background:#ffdacc; textcolor:#000"| 11
| ''[[A Haunted House (film)|A Haunted House]]'' || [[Open Road Films]] || Michael Tiddes
(director); [[Marlon Wayans]], Rick Alvarez (screenplay); Marlon Wayans, [[Nick Swardson]],
[[David Koechner]], [[Essence Atkins]], [[Cedric the Entertainer]], [[Alanna Ubach]], Jamie
Noel || [[Comedy]], [[Horror film|Horror]] || <center><ref>{{cite web |
url=http://www.comingsoon.net/films.php?id=87046 | title=A Haunted House |
publisher=ComingSoon.net | accessdate=2012-10-06}}</ref></center>
|-
| ''[[Gangster Squad]]'' || [[Warner Bros.]] || [[Ruben Fleischer]] (director); Will Beall
(screenplay); [[Ryan Gosling]], [[Emma Stone]], [[Sean Penn]], [[Josh Brolin]], [[Robert
Patrick]], [[Michael Peña]], [[Giovanni Ribisi]], [[Anthony Mackie]], [[Nick Nolte]], [[Wade
Williams]], [[Mireille Enos]], [[Sullivan Stapleton]], [[Frank Grillo]] || [[Action
film|Action]], Crime || <center><ref>{{cite web |
url=http://www.comingsoon.net/films.php?id=51376 | title=Gangster Squad |
publisher=ComingSoon.net | accessdate=2011-09-29}}</ref></center>
|-
| ''[[Matru Ki Bijlee Ka Mandola]]'' || [[Fox Star Studios]] || [[Vishal Bhardwaj]]
(director/screenplay); [[Abhishek Chaubey]] (screenplay); [[Imran Khan (actor)|Imran Khan]],
[[Anushka Sharma]], [[Pankaj Kapur]], [[Shabana Azmi]], [[Arya Babbar]], [[Aamir Khan]],
[[Ajay Devgan]]  || [[Romantic comedy film|Romantic Comedy]] || <center><ref>{{cite web |
url=http://www.boxofficemojo.com/movies/?id=matrukibijleekamandola.htm | title=Matru Ki Bijlee
Ka Mandola | publisher=BoxOfficeMojo.com | accessdate=2013-01-07}}</ref></center>
|-
| ''[[Struck by Lightning (2012 film)|Struck by Lightning]]'' || Tribeca Film || [[Brian
Dannelly]] (director); [[Chris Colfer]] (screenplay); [[Chris Colfer]], [[Allison Janney]],
[[Christina Hendricks]], [[Sarah Hyland]], [[Carter Jenkins]], [[Brad William Henke]], [[Rebel
Wilson]], [[Angela Kinsey]], [[Polly Bergen]], [[Dermot Mulroney]], [[Allie Grant]], [[Ashley
Rickards]], [[Robbie Amell]], [[Charlie Finn]], [[Matt Prokop]], [[Adam Kolkin]] || Comedy,
Drama || <center><ref>{{cite web | url=http://www.comingsoon.net/films.php?id=79818 |
title=Struck by Lightning | publisher=ComingSoon.net | accessdate=2013-01-09}}</ref></center>
|-
| rowspan="1" style="text-align:center; background:#ffdacc; textcolor:#000"| 17
| ''[[Hansel and Gretel: Witch Hunters]]'' || [[Paramount Pictures]] /
[[Metro-Goldwyn-Mayer|MGM]] || [[Tommy Wirkola]] (director/screenplay); [[Dante Harper]]
(screenplay); [[Jeremy Renner]], [[Gemma Arterton]], [[Zoe Bell]], [[Famke Janssen]], [[Peter
Stormare]], [[Ingrid Bolso Berdal]], [[Derek Mears]], [[Thomas Mann (actor)|Thomas Mann]],
[[Pihla Viitala]], [[Robin Atkin Downes]], [[Rainer Bock]], [[Monique Ganderton]], [[Joanna
Kulig]] || [[Action film|Action]], [[Fantasy]] || <center><ref>{{cite web |
url=http://www.hanselandgretelmovie.com/_apps/releasedates/release-dates.html | title=Hansel
and Gretel: Witch Hunters | publisher=hanselandgretelmovie.com |
accessdate=2013-04-28}}</ref></center>
```

```
|-
| rowspan="5" style="text-align:center; background:#ffdacc; textcolor:#000"| 18
| ''[[Broken City (film)|Broken City]]'' || [[20th Century Fox]] || [[Hughes brothers|Allen
Hughes]] (director); Brian Tucker (screenplay); [[Mark Wahlberg]], [[Russell Crowe]],
[[Catherine Zeta-Jones]], [[Barry Pepper]], [[Kyle Chandler]], [[Natalie Martinez]], [[Justin
Chambers]], [[Alona Tal]], [[Jeffrey Wright (actor)|Jeffrey Wright]], Catherine Kim Poon,
[[Justin Chambers]], [[Alona Tal]], James Rawlings, [[James Ransone]], [[Judd Lormand]] ||
Crime || <center><ref>{{cite web | url=http://www.comingsoon.net/films.php?id=46876 |
title=Broken City | publisher=ComingSoon.net | accessdate=2011-10-27}}</ref></center>
|-
| ''[[LUV (film)|LUV]]'' || Indomina Media || Sheldon Candis (director/screenplay); [[Common
(entertainer)|Common]], [[Dennis Haysbert]], [[Danny Glover]], [[Charles S. Dutton]], [[Meagan
Good]] || Drama || <center><ref>{{cite web |
url=http://www.boxofficemojo.com/movies/?id=luv.htm | title=LUV | publisher=BoxOfficeMojo.com
| accessdate=2013-01-30}}</ref></center>
|-
| ''[[Mama (2013 film)|Mama]]'' || [[Universal Studios|Universal Pictures]] || Andres
Muschietti (director/screenplay); [[Neil Cross]], Barbara Muschietti (screenplay); [[Jessica
Chastain]], [[Nikolaj Coster-Waldau]], [[Daniel Kash]], [[David Fox (actor)|David Fox]] ||
[[Horror film|Horror]], [[Thriller film|Thriller]] || <center><ref>{{cite web |
url=http://www.comingsoon.net/films.php?id=80014 | title=Mama | publisher=ComingSoon.net |
accessdate=2012-08-12}}</ref></center>
|-
| ''[[Officer Down]]'' || [[Anchor Bay Films]] || Brian A Miller (director); John Chase
(screenplay); [[Stephen Dorff]], [[Dominic Purcell]], [[David Boreanaz]], [[Soulja Boy]],
[[Stephen Lang (actor)|Stephen Lang]], [[James Woods]], [[Walton Goggins]], [[Tommy Flanagan
(actor)|Tommy Flanagan]], [[Oleg Taktarov]], [[Johnny Messner (actor)|Johnny Messner]] ||
Action || <center><ref>{{cite web |
url=http://www.boxofficemojo.com/movies/?id=officerdown.htm | title=Officer Down |
publisher=BoxOfficerMojo.com | accessdate=2013-01-30}}</ref></center>
|-
| ''[[The Last Stand (2013 film)|The Last Stand]]'' || [[Lions Gate Entertainment|Lionsgate]]
|| [[Kim Ji-woon]] (director); Andrew Knauer, [[Jeffrey Nachmanoff]] (screenplay); [[Arnold
Schwarzenegger]], [[Forest Whitaker]], [[Peter Stormare]], [[Jaimie Alexander]], [[Rodrigo
Santoro]], [[Zach Gilford]], [[Johnny Knoxville]], [[Harry Dean Stanton]], [[Luis Guzmán]],
[[Génesis Rodríguez]], [[Daniel Henney]] || [[Action film|Action]] || <center><ref>{{cite web
| url=http://www.comingsoon.net/films.php?id=76103 | title=The Last Stand |
publisher=ComingSoon.net | accessdate=2011-07-20}}</ref></center>
|-
| rowspan="6" style="text-align:center; background:#ffdacc; textcolor:#000"| 25
| ''[[John Dies at the End (film)|John Dies at the End]]'' || [[Magnolia Pictures]] || [[Don
Coscarelli]] (director/screenplay); [[Paul Giamatti]], [[Clancy Brown]], [[Glynn Turman]],
[[Doug Jones (actor)|Doug Jones]], [[Daniel Roebuck]], [[Jimmy Wong]], [[Angus Scrimm]] ||
Comedy, Horror || <center><ref>{{cite web |
url=http://boxofficemojo.com/movies/?id=johndiesattheend.htm | title=John Dies at the End |
publisher=boxofficemojo.com | accessdate=2013-01-07}}</ref></center>
|-
| ''[[Knife Fight]]'' || [[IFC Films]] || [[Bill Guttentag]] (director/screenplay); [[Chris
Lehane]] (screenplay); [[Rob Lowe]], [[Carrie-Anne Moss]], [[Jamie Chung]], [[Richard
Schiff]], [[Eric McCormack]], [[Julie Bowen]], [[Jennifer Morrison]], [[Amanda Crew]],
[[Saffron Burrows]], [[Shirley Manson]], [[Davey Havok]] || [[Political Thriller]] ||
<center><ref>{{cite web | url=http://www.boxofficemojo.com/movies/?id=knifefight.htm |
title=Knife Fight | publisher=boxofficemojo.com | accessdate=2013-01-12}}</ref></center>
|-
| ''[[Movie 43]]'' || [[Relativity Media]] || Patrik Forsberg (director/screenplay);
[[Elizabeth Banks]], [[Steven Brill (scriptwriter)|Steven Brill]], [[Steve Carr]], [[Rusty
Cundieff]], James Duffy, [[Griffin Dunne]], [[Peter Farrelly]], [[James Gunn (filmmaker)|James
```

```
Gunn]], [[Bob Odenkirk]], [[Brett Ratner]], Jonathan van Tulleken (directors); Steve Baker,
Will Carlough, Matt Portenoy, [[Greg Pritikin]], Rocky Russo, Jeremy Sosenko, Elizabeth Wright
Shapiro (screenplay); [[Hugh Jackman]], [[Emma Stone]], [[Gerard Butler]], [[Chloë Grace
Moretz]], [[Elizabeth Banks]], [[Kristen Bell]], [[Naomi Watts]], [[Anna Faris]], [[Chris
Pratt]], [[Kate Winslet]], [[Josh Duhamel]], [[Richard Gere]], [[Uma Thurman]], [[Halle
Berry]], [[Patrick Warburton]], [[Christopher Mintz-Plasse]], [[Liev Schreiber]], [[Seann
William Scott]], [[Jason Sudeikis]], [[Kate Bosworth]], [[Bobby Cannavale]], [[Justin Long]],
[[J. B. Smoove]], [[Kieran Culkin]], [[Leslie Bibb]], [[Terrence Howard]], [[Jack McBrayer]],
[[Tony Shalhoub]], [[Matt Walsh (comedian)|Matt Walsh]], [[Stephen Merchant]], [[Jimmy
Bennett]], [[Emily Alyn Lind]], [[Jeremy Allen White]] || Comedy || <center><ref>{{cite news |
title=Relativity Sets Dates for Raven & Farrelly/Wessler Comedy | publisher=ComingSoon.net |
url=http://www.comingsoon.net/news/movienews.php?id=77135 |
accessdate=2011-04-26}}</ref></center>
|-
| ''[[Parker (2013 film)|Parker]]'' || [[FilmDistrict]] || [[Taylor Hackford]] (director);
John McLaughlin (screenplay); [[Jason Statham]], [[Jennifer Lopez]], [[Bobby Cannavale]],
[[Clifton Collins, Jr.]], [[Wendell Pierce]], [[Michael Chiklis]], [[Nick Nolte]], [[Sala
Baker]] || [[Crime]], [[Action film|Action]] || <center><ref>{{cite web |
url=http://www.comingsoon.net/films.php?id=76591 | title=Parker | publisher=ComingSoon.net |
accessdate=2011-08-06}}</ref></center>
|-
| ''[[Race 2]]'' || [[Tips Music Films]] || [[Abbas-Mustan]] (director); Shiraz Ahmed
(screenplay); [[Anil Kapoor]], [[Saif Ali Khan]], [[Deepika Padukone]], [[John Abraham
(actor)|John Abraham]], [[Jacqueline Fernandez]], [[Ameesha Patel]], [[Rajesh Khattar]],
[[Bipasha Basu]]  || Action || <center><ref>{{cite web |
url=http://www.boxofficemojo.com/movies/?id=race2.htm | title=Race 2 |
publisher=BoxOfficeMojo.com | accessdate=2013-01-12}}</ref></center>
|-
| ''[[Television (film)|Television]]'' || Chabial || [[Mostofa Sarwar Farooki]] (director);
Anisul Haque (screenplay); [[Chanchal Chowdhury]], Shahir Kazi Huda, [[Mosharraf Karim]], Iman
Lee, Nusrat Imroz Tisha || Drama || <center><ref>{{cite web |
url=http://www.imdb.com/title/tt2564706/ | title=Television (2012) - IMDb | publisher=imdb.com
| accessdate=2013-01-31}}</ref></center>
|-
! rowspan="22" style="text-align:center; background:thistle; textcolor:#000;"|F<br />E<br
/>B<br />R<br />U<br />A<br />R<br />Y
| rowspan="5" style="text-align:center; background:#d8d8d8; textcolor:#000"| 1
| ''[[Bullet to the Head]]'' || [[Warner Bros.]] || [[Walter Hill (director)|Walter Hill]]
(director/screenplay); [[Alessandro Camon]] (screenplay); [[Sylvester Stallone]], [[Sung
Kang]], [[Sarah Shahi]], [[Adewale Akinnuoye-Agbaje]], [[Christian Slater]], [[Jason Mamoa]],
[[Jon Seda]], [[Brian Van Holt]] || Crime, [[Action film|Action]] || <center><ref>{{cite web |
url=http://www.comingsoon.net/films.php?id=73900 | title=Bullet to the Head |
publisher=ComingSoon.net | accessdate=2012-04-07}}</ref></center>
|-
| ''[[Sound City (film)|Sound City]]'' || Therapy Studios || [[Dave Grohl]] (director); Mark
Monroe (screenplay);|| Documentary || <center><ref>{{cite web |
url=http://www.boxofficemojo.com/movies/?id=soundcity.htm | title=Sound City |
publisher=BoxOfficeMojo.com | accessdate=2013-02-05}}</ref></center>
|-
| ''[[Stand Up Guys]]'' || [[Lions Gate Entertainment|Lionsgate]] || [[Fisher Stevens]]
(director); Noah Haidle (screenplay); [[Al Pacino]], [[Christopher Walken]], [[Julianna
Margulies]], [[Mark Margolis]], [[Alan Arkin]], [[Bill Burr]] || Crime, Comedy ||
<center><ref>{{cite web | url=http://www.comingsoon.net/films.php?id=56518 | title=Stand Up
Guys | publisher=ComingSoon.net | accessdate=2012-09-05}}</ref></center>
|-
| ''[[The Haunting in Connecticut 2: Ghosts of Georgia]]'' || [[Lions Gate
Entertainment|Lionsgate]] || Tom Elkins (director); Dave Coggeshall (screenplay); [[Chad
```

```
Michael Murray]], [[Abigail Spencer]], [[Katee Sackhoff]], [[Cicely Tyson]], [[Emily Alyn
Lind]] || Horror || <center><ref>{{cite web | url=http://www.comingsoon.net/films.php?id=65200
| title=Haunting in Connecticut 2 | publisher=ComingSoon.net |
accessdate=2013-02-01}}</ref></center>
|-
| ''[[Warm Bodies (film)|Warm Bodies]]'' || [[Summit Entertainment]] || [[Jonathan Levine]]
(director/screenplay); [[Nicholas Hoult]], [[Teresa Palmer]], [[Rob Corddry]], [[John
Malkovich]], [[Analeigh Tipton]], [[Dave Franco]], [[Cory Hardrict]] || [[Comedy]], [[Horror
film|Horror]], [[Romance film|Romance]] || <center><ref>{{cite web |
url=http://www.comingsoon.net/films.php?id=62500 | title=Warm Bodies |
publisher=ComingSoon.net | accessdate=2011-03-12}}</ref></center>
|-
| rowspan="6" style="text-align:center; background:#d8d8d8; textcolor:#000"| 8
| ''[[A Glimpse Inside the Mind of Charles Swan III]]'' || [[A24 Films|A24]] || [[Roman
Coppola]] (director/screenplay); [[Charlie Sheen]], [[Jason Schwartzmann]], [[Bill Murray]],
[[Katheryn Winnick]], [[Patricia Arquette]], [[Aubrey Plaza]], [[Mary Elizabeth Winstead]] ||
Comedy || <center><ref>{{cite news | title=A Gimpse Inside the Mind of Charles Swan III |
publisher=ComingSoon.net | url=http://www.comingsoon.net/films.php?id=81887 |
accessdate=2012-11-27}}</ref></center>
|-
| ''[[I Give It a Year]]'' || [[Working Title Films]] || [[Dan Mazer]] (director/screenplay);
[[Rose Byrne]], [[Anna Faris]], [[Rafe Spall]], [[Simon Baker]], [[Minnie Driver]], [[Stephen
Merchant]] || Comedy || <center><ref>{{cite news | title=I Give It a Year |
publisher=www.bbfc.co.uk | url=http://www.bbfc.co.uk/releases/i-give-it-year-2013-3 |
accessdate=2013-04-18}}</ref></center>
|-
| ''[[Identity Thief]]'' || [[Universal Studios|Universal Pictures]] || [[Seth Gordon]]
(director); [[Steve Conrad]], [[Craig Mazin]] (screenplay); [[Jason Bateman]], [[Melissa
McCarthy]], [[Amanda Peet]], [[John Cho]], [[Clark Duke]], Maggie Elizabeth Jones,
Mary-Charles Jones, [[Jon Favreau]], [[Génesis Rodríguez]], [[Eric Stonestreet]], [[T.I.]],
[[Ben Falcone]], [[Morris Chestnut]], Justin Wheelon || Action, Comedy || <center><ref>{{cite
news | title=Identity Thief | publisher=ComingSoon.net |
url=http://www.comingsoon.net/films.php?id=81067 | accessdate=2012-03-02}}</ref></center>
|-
| ''[[Side Effects (2013 film)|Side Effects]]'' || [[Open Road Films]] || [[Steven
Soderbergh]] (director); [[Scott Z. Burns]] (screenplay); [[Rooney Mara]], [[Jude Law]],
[[Channing Tatum]], [[Catherine Zeta-Jones]], [[Vinessa Shaw]], [[David Costabile]] ||
[[Crime]] || <center><ref>{{cite web | title=The Bitter Pill | publisher=ComingSoon.net |
url=http://www.comingsoon.net/films.php?id=84696 | accessdate=2011-04-16}}</ref></center>
|-
| ''[[Special 26]]'' || [[Viacom 18 Motion Pictures]] || [[Neeraj Pandey]]
(director/screenplay); [[Akshay Kumar]], [[Kajal Aggarwal]], [[Anupam Kher]], [[Jimmy
Shergill]], [[Divya Dutta]], [[Manoj Bajpai]], [[Kharaj Mukherjee]], [[Rajesh Sharma
(actor)|Rajesh Sharma]], [[Kishor Kadam]], [[Deepraj Rana]] || Crime || <center><ref>{{cite
web | title= Special 26 | publisher=http://www.imdb.com |
url=http://www.imdb.com/title/tt2377938/ | accessdate=2013-03-09}}</ref></center>
|-
| ''[[Top Gun|Top Gun 3D]]'' || [[Paramount Pictures]] || [[Tony Scott]] (director); [[Jim
Cash]], [[Jack Epps Jr.]] (screenplay); [[Tom Cruise]], [[Kelly McGillis]], [[Val Kilmer]],
[[James Tolkan]], [[Tom Skerritt]], [[Rick Rossovich]], [[Michael Ironside]], [[John Stockwell
(actor)|John Stockwell]], [[Tim Robbins]], [[Whip Hubley]], [[Barry Tubb]], [[David L.
Paterson|David Patterson]], [[Adrian Pasdar]], [[Clarence Gilyard, Jr.]], [[Duke Stroud]],
[[Linda Rae Jurgens]] || [[Action film|Action]] || <center><ref>{{cite web | title=Top Gun 3D
| publisher=ComingSoon.net | url=http://www.comingsoon.net/films.php?id=97961 |
accessdate=2011-04-16}}</ref></center>
|-
| style="text-align:center; background:#d8d8d8; textcolor:#000;"| 10
```

```
| ''[[Journey to the West: Conquering the Demons]]'' || [[Huayi Brothers]] / [[China Film Group]] || [[Stephen Chow]], [[Derek Kwok]] (director); [[Huang Bo]], [[Shu Qi]], [[Wen Zhang]], [[Show Luo]], [[Chrissie Chau]] || Action, Comedy || <center><ref>{{cite web | url=http://www.hollywoodreporter.com/news/china-box-office-stephen-chow-420299 | title=Journey to the West | publisher=HollywoodReporter.com | accessdate=2013-02-22}}</ref></center>
|-
| rowspan="4" style="text-align:center; background:#d8d8d8; textcolor:#000"| 14
| ''[[A Good Day to Die Hard]]'' || [[20th Century Fox]] || [[John Moore (director)|John Moore]] (director); [[Skip Woods]] (screenplay); [[Bruce Willis]], [[Jai Courtney]], [[Sebastian Koch]], [[Yuliya Snigir]], Radivoje Bukvić, [[Cole Hauser]], [[Amaury Nolasco]], [[Mary Elizabeth Winstead]] || [[Action film|Action]], [[Thriller (genre)|Thriller]] || <center><ref>{{cite web | url=http://www.comingsoon.net/films.php?id=63534 | title=A Good Day to Die Hard  | publisher=ComingSoon.net | accessdate=2011-10-12}}</ref></center>
|-
| ''[[Beautiful Creatures (2013 film)|Beautiful Creatures]]'' || [[Warner Bros.]] || [[Richard LaGravenese]] (director/screenplay); [[Viola Davis]], [[Alice Englert]], [[Emma Thompson]], [[Emmy Rossum]], [[Jeremy Irons]], [[Alden Ehrenreich]], [[Thomas Mann]], [[Kyle Gallner]] || [[Fantasy]], [[Romance film|Romance]], [[Teen film|Teen]] || <center><ref>{{cite web | url=http://www.comingsoon.net/films.php?id=86589 | title=Beautiful Creatures | publisher=ComingSoon.net | accessdate=2012-02-15}}</ref></center>
|-
| ''[[Murder 3]]'' || [[Fox Star Studios]] || Vishesh Bhatt (director/screenplay); [[Randeep Hooda]], [[Aditi Rao Hydari]], [[Sara Loren]] || Thriller || <center><ref>{{cite web | url=http://www.boxofficemojo.com/movies/?id=murder3.htm | title=Murder 3  | publisher=boxofficemojo.com | accessdate=2013-01-24}}</ref></center>
|-
| ''[[Safe Haven (film)|Safe Haven]]'' || [[Relativity Media]] || [[Lasse Hallström]] (director); [[Dana Stevens (screenwriter)|Dana Stevens]], [[Leslie Bohem]] (screenplay); [[Josh Duhamel]], [[Julianne Hough]], [[David Lyons (actor)|David Lyons]], [[Cobie Smulders]], Mimi Kirkland, Noah Lomax, [[Mike Pniewski]] || Drama, [[Romance film|Romance]] ||<center><ref>{{cite web | title=Safe Haven | publisher=ComingSoon.net | url=http://www.comingsoon.net/films.php?id=68597 | accessdate=2011-02-02}}</ref></center>
|-
| style="text-align:center; background:#d8d8d8; text-color:#000;"| 15
| ''[[Escape from Planet Earth]]'' || [[The Weinstein Company]] || Cal Brunker (director/screenplay); [[Bob Barlen]] (screenplay); [[Brendan Fraser]], [[Rob Corddry]], [[Jessica Alba]], [[William Shatner]], [[Sarah Jessica Parker]], [[Jane Lynch]], [[Sofía Vergara]], [[George Lopez]], [[Craig Robinson (actor)|Craig Robinson]] || Comedy, Family, [[Sci-Fi]] || <center><ref>{{cite web | url=http://www.comingsoon.net/films.php?id=80575 | title=Escape from Planet Earth | publisher=ComingSoon.net | accessdate=2012-01-19}}</ref></center>
|-
| rowspan="5" style="text-align:center; background:#d8d8d8; text-color:#000"| 22
| ''[[Bless Me, Ultima (film)|Bless Me, Ultima]]'' || Arenas Entertainment || [[Carl Franklin]] (director/screenplay); [[Míriam Colón]], [[Benito Martinez (actor)|Benito Martinez]], [[Dolores Heredia]], [[Cástulo Guerra]] || Drama || <center><ref>{{cite web | url=http://www.boxofficemojo.com/movies/?id=blessmeultima.htm | title=Bless Me, Ultima | publisher=BoxOfficeMojo.com | accessdate=2013-02-24}}</ref></center>
|-
| ''[[Dark Skies (film)|Dark Skies]]'' || [[Dimension Films]] || [[Scott Charles Stewart]] (director/screenplay); [[Keri Russell]], [[Josh Hamilton (actor)|Josh Hamilton]], [[Dakota Goyo]] || [[Horror film|Horror]], [[Science fiction film|Sci-Fi]] || <center><ref>{{cite web | url=http://www.comingsoon.net/films.php?id=89656 | title=Dark Skies | publisher=ComingSoon.net | accessdate=2012-09-19}}</ref></center>
|-
| ''[[Kai Po Che!]]'' || [[UTV Motion Pictures]] || [[Abhishek Kapoor]] (director/screenplay); [[Chetan Bhagat]] (screenplay); [[Sushant Singh Rajput]], [[Amit Sadh]], [[Raj Yadav]],
```

```
[[Amrita Puri]], [[Ajay Jadeja]] || Drama, Musical || <center><ref>{{cite web |
url=http://www.boxofficemojo.com/movies/?id=kaipoche.htm | title=Kai Po Che! |
publisher=boxofficemojo.com | accessdate=2013-01-24}}</ref></center>
|-
| ''[[Snitch (film)|Snitch]]'' || [[Summit Entertainment]] || Ric Roman Waugh
(director/screenplay); [[Justin Haythe]] (screenplay); [[Dwayne Johnson]], [[Susan Sarandon]],
[[Benjamin Bratt]], [[Barry Pepper]], [[Jon Bernthal]], [[Michael K. Williams]], [[Melina
Kanakaredes]], [[Nadine Velazquez]], [[Rafi Gavron]], [[David Harbour]] || [[Action
film|Action]] || <center><ref>{{cite web | url=http://www.comingsoon.net/films.php?id=22960 |
title=Snitch | publisher=ComingSoon.net | accessdate=2012-09-19}}</ref></center>
|-
| ''[[To the Wonder]]'' || [[Magnolia Pictures]] || [[Terrence Malick]] (director/screenplay);
[[Ben Affleck]], [[Olga Kurylenko]], [[Rachel McAdams]], [[Javier Bardem]], Charles Baker,
Romina Mondello || [[Romantic drama]] || <center><ref>{{cite web |
url=http://www.odeon.co.uk/fanatic/film_info/m14342/To_The_Wonder/ | title=To the Wonder |
publisher=Odeon.co.uk | accessdate=2013-04-13}}</ref></center>
|-
! rowspan="29" style="text-align:center; background:#98fb98; textcolor:#000;"|M<br />A<br
/>R<br />C<br />H
| rowspan="7" style="text-align:center; background:#e0f9e0; textcolor:#000;"| 1
| ''[[21 & Over (film)|21 & Over]]'' || [[Relativity Media]] || [[Jon Lucas]], [[Scott Moore
(screenwriter)|Scott Moore]] (director/screenplay); [[Miles Teller]], [[Justin Chon]],
[[Skylar Astin]], [[Sarah Wright]], [[François Chau]], [[Jonathan Keltz]], [[Daniel Booko]],
Dustin Ybarra || Comedy || <center><ref>{{cite web |
url=http://www.comingsoon.net/films.php?id=78125 | title=21 and Over |
publisher=ComingSoon.net | accessdate=2012-04-09}}</ref></center>
|-
```

## Our Code for this file: -

```
void filmprocess(string path){
	file f;
	file temp;
	string del;

	f=fopen(path,"r");
	temp=fopen("filmresult.txt","w");
	del="[']";

	while(!feof(f)){
		fgetc(f);
		searchPattern(f,"| ''[[");
		extractBefore(f,temp,"||",del);
		extractBefore(f,temp,"||",del);
		extractBefore(f,temp,"(",del);
		searchPattern(f,"| publisher=");
		extractBefore(f,temp," ",del);
		fputs("\n",temp);
	}
```

```
}

int main()
{
	filmprocess("2013films.txt");
	return 0;
}
```

## Output file (Small snapshot): -

```
A Dark Truth ,  Magnolia Pictures ,  Damian Lee , ComingSoon.net,
Table No. 21 ,  Eros International ,  Aditya Datt , BoxOfficeMojo.com,
Texas Chainsaw 3D ,  Lions Gate Entertainment|Lionsgate ,  John Luessenhop , ComingSoon.net,
The Grandmaster (film)|The Grandmaster ,  Block 2 Pictures / Jet Tone Films ,  Wong Kar-wai ,
Berlinale.de,
A Haunted House (film)|A Haunted House ,  Open Road Films ,  Michael Tiddes , ComingSoon.net,
Gangster Squad ,  Warner Bros. ,  Ruben Fleischer , ComingSoon.net,
Matru Ki Bijlee Ka Mandola ,  Fox Star Studios ,  Vishal Bhardwaj , BoxOfficeMojo.com,
Struck by Lightning (2012 film)|Struck by Lightning ,  Tribeca Film ,  Brian Dannelly ,
ComingSoon.net,
Hansel and Gretel: Witch Hunters ,  Paramount Pictures / Metro-Goldwyn-Mayer|MGM ,  Tommy
Wirkola , hanselandgretelmovie.com,
Broken City (film)|Broken City ,  20th Century Fox ,  Hughes brothers|Allen Hughes ,
ComingSoon.net,
LUV (film)|LUV ,  Indomina Media ,  Sheldon Candis , BoxOfficeMojo.com,
Mama (2013 film)|Mama ,  Universal Studios|Universal Pictures ,  Andres Muschietti ,
ComingSoon.net,
Officer Down ,  Anchor Bay Films ,  Brian A Miller , BoxOfficerMojo.com,
The Last Stand (2013 film)|The Last Stand ,  Lions Gate Entertainment|Lionsgate ,  Kim Ji-woon
, ComingSoon.net,
John Dies at the End (film)|John Dies at the End ,  Magnolia Pictures ,  Don Coscarelli ,
boxofficemojo.com,
Knife Fight ,  IFC Films ,  Bill Guttentag , boxofficemojo.com,
Movie 43 ,  Relativity Media ,  Patrik Forsberg , ComingSoon.net,
Parker (2013 film)|Parker ,  FilmDistrict ,  Taylor Hackford , ComingSoon.net,
Race 2 ,  Tips Music Films ,  Abbas-Mustan , BoxOfficeMojo.com,
Television (film)|Television ,  Chabial ,  Mostofa Sarwar Farooki , imdb.com,
Bullet to the Head ,  Warner Bros. ,  Walter Hill , ComingSoon.net,
Sound City (film)|Sound City ,  Therapy Studios ,  Dave Grohl , BoxOfficeMojo.com,
Stand Up Guys ,  Lions Gate Entertainment|Lionsgate ,  Fisher Stevens , ComingSoon.net,
The Haunting in Connecticut 2: Ghosts of Georgia ,  Lions Gate Entertainment|Lionsgate ,  Tom
Elkins , ComingSoon.net,
Warm Bodies (film)|Warm Bodies ,  Summit Entertainment ,  Jonathan Levine , ComingSoon.net,
A Glimpse Inside the Mind of Charles Swan III ,  A24 Films|A24 ,  Roman Coppola ,
ComingSoon.net,
I Give It a Year ,  Working Title Films ,  Dan Mazer , www.bbfc.co.uk,
Identity Thief ,  Universal Studios|Universal Pictures ,  Seth Gordon , ComingSoon.net,
Side Effects (2013 film)|Side Effects ,  Open Road Films ,  Steven Soderbergh ,
ComingSoon.net,
Special 26 ,  Viacom 18 Motion Pictures ,  Neeraj Pandey , http://www.imdb.com,
Top Gun|Top Gun 3D ,  Paramount Pictures ,  Tony Scott , ComingSoon.net,
```

Journey to the West: Conquering the Demons , Huayi Brothers / China Film Group , Stephen Chow, Derek Kwok , HollywoodReporter.com,
A Good Day to Die Hard , 20th Century Fox , John Moore , ComingSoon.net,
Beautiful Creatures (2013 film)|Beautiful Creatures , Warner Bros. , Richard LaGravenese , ComingSoon.net,
Murder 3 , Fox Star Studios , Vishesh Bhatt , boxofficemojo.com,
Safe Haven (film)|Safe Haven , Relativity Media , Lasse Hallström , ComingSoon.net,
Escape from Planet Earth , The Weinstein Company , Cal Brunker , ComingSoon.net,
Bless Me, Ultima (film)|Bless Me, Ultima , Arenas Entertainment , Carl Franklin , BoxOfficeMojo.com,
Dark Skies (film)|Dark Skies , Dimension Films , Scott Charles Stewart , ComingSoon.net,
Kai Po Che! , UTV Motion Pictures , Abhishek Kapoor , boxofficemojo.com,
Snitch (film)|Snitch , Summit Entertainment , Ric Roman Waugh , ComingSoon.net,
To the Wonder , Magnolia Pictures , Terrence Malick , Odeon.co.uk,
21 & Over (film)|21 & Over , Relativity Media , Jon Lucas, Scott Moore , ComingSoon.net,
I, Me Aur Main , Reliance Entertainment , Kapil Sharma , bollywoodhungama.com,
Jack the Giant Slayer , Warner Bros. / New Line Cinema , Bryan Singer , ComingSoon.net,
Phantom (2013 film)|Phantom , RCR Media Group , Todd Robinson , ComingSoon.net,
Stoker (film)|Stoker , Fox Searchlight Pictures , Park Chan-wook , ComingSoon.net,
The Attacks of 26/11 , Eros International , Ram Gopal Varma , sulekha.com,
The Last Exorcism Part II , CBS Films , Ed Gass-Donnelly , ComingSoon.net,
Dead Man Down , FilmDistrict , Niels Arden Oplev , ComingSoon.net,
Emperor (film)|Emperor , Lions Gate Entertainment|Lionsgate / Roadside Attractions , Peter Webber , BoxOffice,
Im So Excited (film)|Im So Excited , El Deseo , Pedro Almodóvar , moviepre.com,
Oz the Great and Powerful , Walt Disney Pictures / Joe Roth|Roth Films , Sam Raimi , Deadline.com,
Goddess (2013 film)|Goddess , The Film Company , Mark Lamprell , villagecinemas.com.au,
Spring Breakers , A24 Films|A24 , Harmony Korine , inquisitr.com,
The Call (2013 film)|The Call , TriStar Pictures , Brad Anderson , ComingSoon.net,
The Incredible Burt Wonderstone , Warner Bros. / New Line Cinema , Don Scardino , ComingSoon.net,
Welcome to the Punch , IFC Films , Eran Creevy , odeon.co.uk,
Admission (2013 film)|Admission , Focus Features , Paul Weitz , ComingSoon.net,
Inappropriate Comedy , Freestyle Releasing , Vince Offer , ComingSoon.net,
Love and Honor (2013 film)|Love and Honor , IFC Films , Danny Mooney , BoxOfficeMojo.com,
Olympus Has Fallen , FilmDistrict , Antoine Fuqua , ComingSoon.net,
The Croods , 20th Century Fox / DreamWorks Animation , Chris Sanders , ComingSoon.net,
G.I. Joe: Retaliation , Paramount Pictures / Metro-Goldwyn-Mayer|MGM , Jon Chu , allocine.fr,
Trance (2013 film)|Trance , Fox Searchlight Pictures , Danny Boyle , imdb.com,
Himmatwala (2013 film)|Himmatwala , UTV Motion Pictures , Sajid Khan , boxofficemojo.com,
Temptation: Confessions of a Marriage Counselor , Lions Gate Entertainment|Lionsgate , Tyler Perry , ComingSoon.net,

# 7. Lessons Learned

## Aman Chahar

I have always worked in C and JAVA which deprived me of any experience of functional language. As we had to work in Ocaml I got to know another paradigm of languages. Functional languages are very powerful and elegant that can do lot of work with very less code. Our project involved compiling into LLVM which was another learning experience. Building own language from scratch and compiling into LLVM gave lot of insight about how languages actually work.

Some tips for the project: -

If you have no experience in functional language and your target language (especially if it is LLVM), it will be very steep learning curve. Instead of trying to figure out everything on your own, go to TA who has experience in LLVM. It can be very frustrating when implementing simple data structures like Arrays or Strings also takes huge amount of time.

## Miao Yu

I learnt the programming in Ocaml, which is very different from the regular languages I learnt in the past. I still remember that classic sentence to describe Ocaml: I've never spent such a long time to write such short code that can do so much.

However, as I gradually study on this class, I found Ocaml is quite interesting, especially when I learnt the Lambda expression. As a student major not in computer science, I found this course quite interesting and learnt lot of new concepts from the course.

The target language of LLVM is very challenging. We have to read on the LLVM documentations to make sure each operation is correct and have to make sure the pointers are casted correctly when it comes to strings.

# Sikai Huang

We generally learned 9 parts of knowledge in this course.

The structure of compiler

There are five stages of a compiler combine to translate a high level language to a low level language, generally closer to that of the target computer. Each stage, or sub-process, fulfills a single task and has one or more classic techniques for implementation.
Lexical Analyzer: Analyzes the source code; Removes space and comments; Formats it for easy access; Tags language elements with type information; Begins to fill in information in the SYMBOL TABLE. Here in Lexical Analyzer, it uses techniques such as linear expressions, finite state machines and lex.

Syntactic Analyzer: Analyzes the tokenized code for structure; amalgamates symbols into syntactic groups; tags groups with type information. Here it uses techniques such as top-down analyzers, bottom-up analyzers and expression analyzers

Semantic Analyzer: Analyzes the parsed code for meaning; Fills in assumed or missing information; Tags groups with meaning information. It uses attribute grammars.

Code generator: Linearizes the qualified code and produces the equivalent object code. Code generation generally completed by hand-written code

Optimizer: Examines the object code to determine whether there are more efficient of execution. We learned operator reduction here.

Scanning
In this part, we learned language and regular expression first which is important to design our own language, Then we learned about the NFA and DFA.

Parsing
We learned to avoid ambiguity when design language.

Build LR automation and SLR parsing table
LR(0) is easy to learn but it cannot be practical use with a very limited set of grammars. SLR is also very simple. We have to remember their operating principle rather than only their steps.

Types
We learned some language types such as C and Java.

This is an impressive experience in PLT class. Not only I have learned some fantasy knowledge, but also I meet many friends here. This is my first time studying abroad and I met some difficulties at the beginning. However, my teammate never abandon me. They help me a lot in this course.

Code generation
I regard it as the final phase of compilation. We learned optimization; linking and function library

here.

Prolog
I learned the execution as well as its environment and some searching language.

## Baokun Cheng

I tried to understand how these five files, including scanner, parser, ast, cogen, semat work. Then I learned how to define a datatype, an operation, and a function. It is more than Ocaml. We must keep learning some functions and data types in LLVM. My primary work is to design the library function. The funny thing is that when we are declaring a function, we can call many LLVM built in functions, which makes our work easier. I enjoyed the process working with my team members. We had a great time.

## Weiduo Sun

I have learned about basic compiler design such as scanning, parsing, abstract syntax trees, exception throw. Some interesting design such like λ calculus. After learn those things I have some preliminary compiler idea of mine.
I even try to design some little functions since I had little experience. There is a challenge for me to learn Ocaml. But the challenge helps me a lot to understanding more about CS and extend my horizon.

# 8. Appendix

## Scanner.mll

```
(* Ocamllex scanner for MicroC *)

{ open Parser

 let unescape s =
       Scanf.sscanf ("\"" ^ s ^ "\"") "%S%!" (fun x -> x)

 }




let alpha = ['a'-'z' 'A'-'Z']
let hex = ['0'-'9' 'a'-'f' 'A'-'F']
let escape = '\\' (['\\' ''' '"' 'n' 'r' 't']|'x'hex hex)
let escape_char = ''' (escape) '''
let ascii = ([' '-'!' '#'-'[' ']'-'~'])
let digit = ['0'-'9']
let id = alpha (alpha | digit | '_')*
let string = '"' ( (ascii | escape)* as s) '"'
let char = ''' ( ascii | digit ) '''
let float = (digit+) ['.'] digit+
let int = digit+
let whitespace = [' ' '\t' '\r']
let return = '\n'


rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "//"     { comment2 lexbuf }             (* Comments *)
| "/*"     { comment lexbuf }           (* Comments *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| ';'      { SEMI }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
```

```
| '='      { ASSIGN }
| '[' { LBRACKET }
| ']' { RBRACKET }
| "++"      {SPLUS}
| "--"         {SMINUS}
| "=="     { 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 }
| "char"   { CHAR }
| "void"   { VOID }
| "true"   { TRUE }
| "false"  { FALSE }
| "file"   {STRING }
| "string" { STRING }
| "include"     { INCLUDE }
| "new"    { NEW}
| "null"   { NULL}
| ['0'-'9']+ as lxm { LITERAL(int_of_string lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| int as lxm                { LITERAL(int_of_string lxm) }
| float as lxm              { FLOAT_LITERAL(float_of_string lxm) }
| char as lxm               { CHAR_LITERAL( String.get lxm 1 ) }
| escape_char as lxm{ CHAR_LITERAL( String.get (unescape lxm) 1) }
| string                    { STRING_LITERAL(unescape s) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }
and comment2= parse "\n"{token lexbuf}
| _ { comment2 lexbuf}

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

# Parser.mly

```
/* Ocamlyacc parser for BMWSA */

%{
open Ast
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA RBRACKET LBRACKET INCLUDE
%token PLUS MINUS TIMES DIVIDE ASSIGN NOT SPLUS SMINUS NULL
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
%token RETURN IF ELSE FOR WHILE INT BOOL VOID FLOAT CHAR STRING NEW
%token <int> LITERAL
%token <float> FLOAT_LITERAL
%token <string> STRING_LITERAL
%token <string> ID
%token <char> CHAR_LITERAL
%token EOF

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

```
%start program
%type <Ast.program> program

%%

program:
    includes decls EOF { Program($1,$2) }

/*****************
  INCLUDE
*****************/

includes:
    /* nothing */ { [] }
    |   include_list  { List.rev $1 }

include_list:
      include_decl              { [$1] }
    |   include_list include_decl { $2::$1 }

include_decl:
  INCLUDE LPAREN STRING_LITERAL RPAREN SEMI { Include($3) }


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

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

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

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

typ:
    INT { Int }
  | BOOL { Bool }
  | VOID { Void }
  | FLOAT { Float }
```

```
  | CHAR { Char }
  | STRING {String_t}
  | INT TIMES { Intptr}
  | STRING TIMES { String_p }



brackets:
        { 1 }
        | brackets RBRACKET LBRACKET {$1 + 1}


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

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


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

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



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

expr:
    LITERAL          { Literal($1) }
  | STRING_LITERAL   { String_Lit($1) }
  | CHAR_LITERAL     { Char_Lit($1) }
  | FLOAT_LITERAL     {Float_Lit($1) }
  | TRUE             { BoolLit(true) }
  | FALSE            { BoolLit(false) }
  | ID               { Id($1) }
```

```
  | ID SPLUS         { Binop(Assign($1,Binop(Ast.Id($1), Add,
Ast.Literal(1))),Sub,Ast.Literal(1))}
  | ID SMINUS        { Binop(Assign($1,Binop(Ast.Id($1), Sub,
Ast.Literal(1))),Add,Ast.Literal(1))}
  | SMINUS ID        { Assign($2,Binop(Ast.Id($2), Sub, Ast.Literal(1)))}
  | SPLUS ID         { Assign($2,Binop(Ast.Id($2), Add, Ast.Literal(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 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) }
  | MINUS expr %prec NEG { Unop(Neg, $2) }
  | NOT expr         { Unop(Not, $2) }
  | ID ASSIGN expr   { Assign($1, $3) }
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | LPAREN expr RPAREN { $2 }
  | ID LBRACKET expr  RBRACKET { Ary($1,$3) }
  | ID LBRACKET expr  RBRACKET ASSIGN expr { Aryasn($1,$3,$6) }
  | ID ASSIGN NEW LBRACKET expr RBRACKET { Init($1,$5) }
  | NULL LPAREN expr RPAREN { Null($3) }


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

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

# AST

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

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

type uop = Neg | Not
```

```
type typ = Int | Bool | Void | Char | Float | String_t | Intptr |String_p

type dtype = Arraytype of typ * int | Dtype of typ

type bind = typ * string

type expr =
    Literal of int
  | Float_Lit of float
  | String_Lit of string
  | Char_Lit of char
  | BoolLit of bool
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of string * expr
  | Call of string * expr list
  | Ary of string* expr
  | Aryasn of string* expr* expr
  | Init of string * expr
  | Null of expr
  | Noexpr

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

type include_stmt = Include of string

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

type decls_val = bind list * func_decl list

type program =  Program of include_stmt list * decls_val

(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
```

```
  | Mult -> "*"
  | Div -> "/"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"

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

let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | String_Lit(s) -> "" ^ s
  | Id(s) -> s
  | Float_Lit(s) -> string_of_float s
  | Char_Lit(s) -> Char.escaped s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Noexpr -> ""
  | Ary(v,e) -> v ^ "[" ^ string_of_expr e ^ "]"
  | Aryasn(v,e1,e2) -> v ^ "[" ^ string_of_expr e1 ^ "]" ^ string_of_expr e2
  | Init(v,e) -> v ^ "=" ^ "new" ^ "[" ^ string_of_expr e ^ "]"
  | Null(e) -> "null" ^ "(" ^ string_of_expr e ^ ")"

let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | For(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
      string_of_expr e3  ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s

let string_of_typ = function
    Int -> "int"
```

```
  | Bool -> "bool"
  | Void -> "void"
  | String_t -> "string"
  | Float -> "float"
  | Char -> "char"
  | Intptr -> "int *"
  | String_p -> "string *"

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

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

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

# Bmwsa

```
(* Top-level of the BMWSA compiler: scan & parse the input,
   check the resulting AST, generate LLVM IR, and dump the module *)

type action = Ast | LLVM_IR | Compile

let _ =
  let action = if Array.length Sys.argv > 1 then
    List.assoc Sys.argv.(1) [ ("-a", Ast);  (* Print the AST only *)
                              ("-l", LLVM_IR);  (* Generate LLVM, don't check *)
                              ("-c", Compile) ] (* Generate, check LLVM IR *)
  else Compile in
  (* let file_in  = open_in "stdlib.bmwsa" in  *)
  let lexbuf = Lexing.from_channel stdin in
  let ast = Parser.program Scanner.token lexbuf in
  (* Semant.initial_check ast; *)
    Semant.initial_check ast
  ;

  let file_in  = open_in "stdlib.bmwsa" in
  let lexbuf2 = Lexing.from_channel file_in in
  let ast2 = Parser.program Scanner.token lexbuf2 in
  Semant.initial_check ast;
(* (try Semant.initial_check ast
```

```
  with _->print_string(";sbsbsbsbsbsbsb\n")); *)
  match action with
    Ast -> print_string (Ast.string_of_program ast)
  | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate (ast,ast2) ))
  | Compile -> let m = Codegen.translate (ast,ast2) in
    Llvm_analysis.assert_valid_module m;
    print_string (Llvm.string_of_llmodule m)
```

# Codegen

```
(* 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/

*)
(*open Printf*)

module L = Llvm
module A = Ast

module StringMap = Map.Make(String)
```

```
let rec translate (A.Program(first, second), A.Program(first1,second1)) =
  (* let b = process_includes first in  *)
  let (globals,functions) = second
  and (globals2,functions2) = second1 in

  let functions = List.append functions2 functions in
  (* let translate (globals,functions) = *)
  let context = L.global_context () in
```

```
let the_module = L.create_module context "BMWSA"

and i64_t  = L.i64_type  context
and i32_t  = L.i32_type  context
and i8_t   = L.i8_type   context
and f_t    =  L.double_type context
and i1_t   = L.i1_type   context
(* and str_typ = Arraytype(Char,1)  *)
and void_t = L.void_type context in
let str_t = L.pointer_type i8_t
and void_ptr=L.pointer_type i32_t in


let ltype_of_typ = function
    A.Int -> i32_t
  | A.Bool -> i1_t
  | A.Void -> void_t
  | A.Float -> f_t
  | A.Char -> i8_t
  | A.String_t -> str_t
  | A.Intptr -> L.pointer_type i32_t
  | A.String_p -> L.pointer_type str_t in
(* Declare each global variable; remember its value in a map *)
let global_vars =
  let global_var m (t, n) =
    let init = L.const_null (ltype_of_typ t)
    in StringMap.add n (L.define_global n init the_module) m in
  List.fold_left global_var StringMap.empty globals in

(* Declare printf(), which the print built-in function will call *)
let printf_t = (L.var_arg_function_type i32_t [| L.pointer_type i8_t |])
and fopen_t=(L.function_type str_t [|str_t;str_t|])
and fputs_t=(L.function_type str_t [|str_t;str_t|])
and fseek_t=(L.function_type str_t [|str_t;i64_t;i32_t|])
and ftell_t=(L.function_type i64_t [|str_t|])
and fgetc_t=(L.function_type i8_t [|str_t|])
and feof_t=(L.function_type i1_t [|str_t|])
and fputc_t=(L.function_type i8_t [|i8_t;str_t|])
and fremove_t=(L.function_type i32_t [|str_t|])
and frename_t=(L.function_type i32_t [|str_t;str_t|])
and memcpy_t = (L.function_type i8_t [|str_t;str_t;i32_t|])
and malloc_t = (L.function_type str_t [|i32_t|])
and calloc_t=(L.function_type str_t [|i32_t;i32_t|])
and itoa_t=(L.function_type i32_t [|str_t|])
and free_t=(L.function_type str_t [|str_t|])
and sprintf_t= (L.var_arg_function_type str_t [|str_t;L.pointer_type i8_t|])
and strcp_t= (L.function_type i32_t [|str_t;str_t|])
and atoi_t = (L.function_type i32_t [|str_t|])
and strlen_t = (L.function_type i32_t [|str_t|])
and fgets_t = (L.function_type str_t [|str_t;i32_t;str_t|])
```

```
in
  let printf_func = L.declare_function "printf" printf_t the_module
  and feof_fun=L.declare_function "feof" feof_t the_module
  and fgetc_fun=L.declare_function "fgetc" fgetc_t the_module
  and fopen_fun=L.declare_function "fopen" fopen_t the_module
  and fputs_fun=L.declare_function "fputs" fputs_t the_module
  and fseek_fun=L.declare_function "fseek" fseek_t the_module
  and ftell_fun=L.declare_function "ftell" ftell_t the_module
  and fputc_fun=L.declare_function "fputc" fputc_t the_module
  and fremove_fun=L.declare_function "remove" fremove_t the_module
  and frename_fun=L.declare_function "rename" frename_t the_module
  and memcpy_fun = L.declare_function "memcpy" memcpy_t the_module
  and malloc_fun = L.declare_function "malloc" malloc_t the_module
  and calloc_fun=L.declare_function "calloc" calloc_t the_module
  and strtok_fun=L.declare_function "strtok" fopen_t the_module
  and itoa_fun = L.declare_function "itoa" itoa_t the_module
  and free_fun = L.declare_function "free" free_t the_module
  and sprintf_fun = L.declare_function "sprintf" sprintf_t the_module
  and strcp_fun= L.declare_function "strcmp" strcp_t the_module
  and fatoi_fun = L.declare_function "atoi" atoi_t the_module
  and fstrlen_fun = L.declare_function "strlen" strlen_t the_module
  and fgets_fun = L.declare_function "fgets" fgets_t the_module

in


  (* Define each function (arguments and return type) so we can call it *)
let  function_decls =
    let function_decl m fdecl =
      let name = fdecl.A.fname
      and formal_types = Array.of_list (List.map (fun (t,_) -> ltype_of_typ t)
fdecl.A.formals)
      in

      let ftype = L.function_type (ltype_of_typ fdecl.A.typ) formal_types in


      StringMap.add name (L.define_function name ftype the_module, fdecl) m in


    List.fold_left function_decl StringMap.empty functions 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.A.fname 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 in
    let str_format_str = L.build_global_stringptr "%s\n" "fmt2" builder in
    let chr_format_str = L.build_global_stringptr "%c\n" "fmt3" builder in
    let read_str= L.build_global_stringptr "a" "fmt4" 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 in

      let 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.A.formals
          (Array.to_list (L.params the_function)) in
      List.fold_left add_local formals fdecl.A.locals in

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

and codegen_string_build e builder =

  let s = L.build_global_stringptr e "tmp1" builder in
  (* try commenting these two lines and compare the result *)
   let zero = L.const_int i32_t 0 in
   L.build_in_bounds_gep s [| zero |] "tmp1" builder


and  codegen_print e builder =
(* print_string "reached here" in *)
  let s = L.build_global_stringptr e "tmp1" builder in
  (* try commenting these two lines and compare the result *)
   let zero = L.const_int i32_t 0 in
  let s = L.build_in_bounds_gep s [| zero |] "tmp1" builder in
    L.build_call printf_func [| s |] "printf" builder in

(* let rec string_expr builder = function

 A.String_Lit s -> s
 | A.Binop (e1, op, e2) ->
     let e1' = string_expr builder e1
     and e2' = string_expr builder e2 in
     let str_concat =  e1' ^ e2'
 in
    codegen_string_build str_concat builder
```

```
in *)

let rec expr builder = function
       A.Literal i -> L.const_int i32_t i
     | A.BoolLit b -> L.const_int i1_t (if b then 1 else 0)
     | A.String_Lit s -> codegen_string_build s builder
     | A.Float_Lit f -> L.const_float f_t f
     | A.Char_Lit c -> L.const_int i8_t (Char.code c)
      | A.Noexpr -> L.const_int i32_t 0
      | A.Id s -> L.build_load (lookup s) s builder

        | A.Ary(e1, e2) -> let para1=(expr builder (A.Id e1))
  and para2=(expr builder e2) in
  let k=L.build_in_bounds_gep para1 [|para2|] "tmpp" builder in
  L.build_load k "deref" builder
  | A.Aryasn(e1, e2,e3) -> let para1=(expr builder (A.Id e1))
  and para2=(expr builder e2)
  and para3=(expr builder e3)
  in let k=L.build_in_bounds_gep para1 [|para2|] "tmpp" builder in
  L.build_store para3 k builder
  |   A.Init(e1,e2) -> let cnt1= (lookup e1) and cnt2= expr builder e2 in
  let tp=L.element_type (L.type_of cnt1) in
  let sz=L.size_of tp in
  let sz1=L.build_intcast sz (i32_t) "intc" builder in
  let dt=L.build_bitcast (L.build_call calloc_fun [|cnt2;sz1|] "tmpa" builder) tp "tmpb"
builder in
  L.build_store dt cnt1 builder
  | A.Null(e1) -> let cnt1= expr builder e1 in
  L.build_is_null cnt1 "isnull" builder

  (*We allow the add between a float and an integer, the type is determined by the first
term*)

      | A.Binop (e1, op, e2) ->
         let e1'=(expr builder e1) and tp1=L.type_of(L.const_null i32_t)
and tp2=(L.type_of (L.const_null f_t)) and e22= (expr builder e2)
in (let tp3=L.type_of (e1') and tp4=L.type_of(e22)in
  (let e2'=
  (if tp3<>tp4 then (if tp3=tp2
   then L.build_sitofp e22 tp3 "int2fp" builder
   else L.build_fptosi e22 tp3 "fp2int" builder)
   else e22) in
      (match op with
        A.Add     -> (if tp3=tp2 then (L.build_fadd) else (L.build_add))
      | A.Sub     -> (if tp3=tp2 then (L.build_fsub) else (L.build_sub))
      | A.Mult    -> (if tp3=tp2 then (L.build_fmul) else (L.build_mul))
      | A.Div     -> (if tp3=tp2 then (L.build_fdiv) else (L.build_sdiv))
      | 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
      ) e1' e2' "tmp" builder))
      | A.Unop(op, e) ->
         let e' = expr builder e in
         (match op with
           A.Neg     -> L.build_neg
         | A.Not     -> L.build_not) e' "tmp" builder
      | A.Assign (s, e) -> let e' = expr builder e in
                           ignore (L.build_store e' (lookup s) builder); e'
| A.Call ("print", [e])  ->
      L.build_call printf_func [| int_format_str ;  (expr builder e) |]
      "printf" builder
  | A.Call ("printstring", [e]) -> L.build_call printf_func [| str_format_str ; (expr builder
e) |] "tmp1" builder

| A.Call ("printx", e)  ->
      let actuals = List.rev (List.map (expr builder) (List.rev e)) in
  L.build_call printf_func (Array.of_list actuals) "tmp1" builder
  | A.Call("strtok",e)->
      let actuals = List.rev (List.map (expr builder) (List.rev e)) in
  L.build_call strtok_fun (Array.of_list actuals) "tmpx" builder
  | A.Call("free",[e])->
      let actual = expr builder e in
  L.build_call free_fun [|(L.build_bitcast actual str_t "strt" builder)|] "tmps" builder

  | A.Call("lengthOf",e) ->
     let actuals = List.rev (List.map (expr builder) (List.rev e)) in
  L.build_call fstrlen_fun (Array.of_list actuals) "tmpx" builder
  | A.Call("sprintff",e) ->
     let actuals = List.rev (List.map (expr builder) (List.rev e)) in
  L.build_call sprintf_fun (Array.of_list actuals) "tmpx" builder
  | A.Call("strcmp",e) ->
     let actuals = List.rev (List.map (expr builder) (List.rev e)) in
  L.build_call strcp_fun (Array.of_list actuals) "tmpx" builder
  | A.Call("fgets",e) ->
     let actuals = List.rev (List.map (expr builder) (List.rev e)) in
  L.build_call fgets_fun (Array.of_list actuals) "tmpx" builder




      | A.Call ("size",[e]) -> let cnt=expr builder e in
      let cnt2=(L.build_call fopen_fun [|cnt;read_str|] "tmp0" builder) in
      L.build_call ftell_fun [|cnt2|] "tmp7" builder
      | A.Call ("feof",[e])-> let cnt=expr builder e in
      L.build_call feof_fun [|cnt|] "temp1" builder
```

```
      | A.Call ("fgetc",[e]) -> let cnt=expr builder e in
  L.build_call fgetc_fun [|cnt|] "temp1" builder
      | A.Call ("remove",[e])-> let cnt=expr builder e in
  L.build_call fremove_fun [|cnt|] "temp1" builder
    | A.Call ("rename",e) -> let actuals = List.rev (List.map (expr builder) (List.rev e)) in
  L.build_call frename_fun (Array.of_list actuals) "tmp2" builder
      | A.Call ("fff", [e;f]) -> let cnt=expr builder e and cnt2=expr builder f in
  L.build_call fputs_fun [|cnt2;(L.build_call fopen_fun [|cnt;read_str|] "tmp0" builder)|]
"tmp5" builder
| A.Call ("memcpy",e) -> let actuals = List.rev (List.map (expr builder) (List.rev e)) in
      L.build_call memcpy_fun (Array.of_list actuals) "tmp8" builder
| A.Call ("malloc", e)-> let actuals = List.rev (List.map (expr builder) (List.rev e)) in
      L.build_call malloc_fun (Array.of_list actuals) "tmp6" builder
      | A.Call ("fopen", e) -> let actuals = List.rev (List.map (expr builder) (List.rev e))
in
  L.build_call fopen_fun (Array.of_list actuals) "tmp2" builder
  | A.Call ("fputc",e) ->let actuals = List.rev (List.map (expr builder) (List.rev e)) in
  L.build_call fputc_fun (Array.of_list actuals) "tmp3" builder
      | A.Call ("fputs",e) ->let actuals = List.rev (List.map (expr builder) (List.rev e)) in

  L.build_call fputs_fun (Array.of_list actuals) "tmp3" builder
  | A.Call ("atoi",e)->let actuals = List.rev (List.map (expr builder) (List.rev e)) in

  L.build_call fatoi_fun (Array.of_list actuals) "tmp3" builder
     | A.Call (f, act) ->
         let (fdef, fdecl) = StringMap.find f function_decls in
        let actuals = List.rev (List.map (expr builder) (List.rev act)) in
        let result = (match fdecl.A.typ with A.Void -> ""
                                           | _ -> f ^ "_result") in
         L.build_call fdef (Array.of_list actuals) result builder
    in

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

    (* Build the code for the given statement; return the builder for
       the statement's successor *)
    let rec stmt builder = function
       A.Block sl -> List.fold_left stmt builder sl
      | A.Expr e -> ignore (expr builder e); builder
      | A.Return e -> ignore (match fdecl.A.typ with
         A.Void -> L.build_ret_void builder
       | _ -> L.build_ret (expr builder e) builder); builder
      | A.If (predicate, then_stmt, else_stmt) ->
         let bool_val = expr builder predicate in
        let merge_bb = L.append_block context "merge" the_function in
```

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

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

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

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

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

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

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

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

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

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

(* let f = List.iter build_function_body functions in   *)
(* let f1 = List.append functions functions2 in  *)
List.iter build_function_body functions;
(* let len = List.length functions in
ignore(print_int len); *)
(* in ignore(); *)

(* and g = process_includes the_module context in ignore(); *)
(* let file_in = open_in "stdlib.bmwsa" in
let lexbuf = Lexing.from_channel stdin in
```

```
  let ast = Parser.program Scanner.token lexbuf in
  ignore(close_in file_in);
  translate ast; *)
  (* match ast with (Program(first,second)) ->  *)
  (* print_string "Reached here" in  *)
  (* let (globals,sfunctions) = second in  *)
  (* List.iter build_function_body functions in *)

  (* List.iter build_function_body sfunctions; *)
  (* let temp_print = List.length globals in ignore(print_int temp_print); *)
  the_module
```

# Semant.ml

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

open Ast

module StringMap = Map.Make(String)

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

   Check each global variable, then check each function *)

let initial_check (Program(program1, decls_val)) =

(* match program1 with Program(includes, decls_val) ->  *)

let (globals, functions) = decls_val in

(* let check (globals,functions )= *)


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

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

```
    | _ -> ()
  in

  (* Raise an exception of the given rvalue type cannot be assigned to
     the given lvalue type *)
  let check_assign lvaluet rvaluet err =
     if lvaluet == rvaluet then lvaluet else raise err
  in

  (**** Checking Global Variables ****)

  List.iter (check_not_void (fun n -> "illegal void global " ^ n)) globals;

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

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

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

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

  (* Function declaration for a named function *)

  let built_in_decls =

      StringMap.add "fopen" { typ = String_t; fname = "fopen"; formals =
[(String_t,"x");(String_t,"x")];
       locals = []; body = [] }
      (StringMap.add "fputs" { typ = String_t; fname = "fputs"; formals =
[(String_t,"x");(String_t,"x")];
             locals = []; body = [] }
            (StringMap.add "printx" { typ = String_t; fname = "printx"; formals =
[(String_t,"x");(String_t,"x")];
             locals = []; body = [] }
           (StringMap.add "print"
           { typ = Void; fname = "print"; formals = [(Int, "x")];
             locals = []; body = [] }
            (StringMap.singleton "printstring"
           {
        typ = Void; fname= "printstring"; formals = [(String_t, "x")];
        locals =[]; body=[] }))))
     (*(StringMap.singleton "printb"
    { typ = Void; fname = "printb"; formals = [(Bool, "x")];
      locals = []; body = [] }) *)
   in

  let function_decls = List.fold_left (fun m fd -> StringMap.add fd.fname fd m)
                        built_in_decls functions
```

```
  in

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

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

  let check_function func =

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

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

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

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

    (* Type of each variable (global, formal, or local *)
    let symbols = List.fold_left (fun m (t, n) -> StringMap.add n t m)
        StringMap.empty (globals @ func.formals @ func.locals )
    in

    let type_of_identifier s =
      try StringMap.find s symbols
      with Not_found -> raise (Failure ("undeclared identifier " ^ s))
    in

    (* Return the type of an expression or throw an exception *)
    let rec expr = function
        Literal _ -> Int
      | BoolLit _ -> Bool
      | String_Lit s -> String_t
      | Float_Lit _ -> Float
      | Char_Lit _ -> Char
      | Id s -> type_of_identifier s
      | Binop(e1, op, e2) as e -> let t1 = expr e1 and t2 = expr e2 in


         (match op with
          Add | Sub -> (match(t1,t2) with (Int,Int) -> Int | (Float,Float) -> Float |
(Float,Int) -> Float | (Int,Float) -> Int | (String_t,String_t) -> String_t | (Char,Char) ->
Char| _->raise (Failure ("illegal binary operator " ^
              string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
              string_of_typ t2 ^ " in " ^ string_of_expr e)) )
```

```
          | Mult | Div -> (match(t1,t2) with (Int,Int)->Int|(Float,Float)->Float|
(Char,Char)->Char|(Float,Int)->Float| (Int,Float)-> Int|_-> raise (Failure ("illegal binary
operator " ^
              string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
              string_of_typ t2 ^ " in " ^ string_of_expr e)) )
          | Equal | Neq when t1 = t2 -> Bool
               | Less | Leq | Greater | Geq when t1 = Int && t2 = Int -> Bool
               | And | Or when t1 = Bool && t2 = Bool -> Bool
          | _ -> raise (Failure ("illegal binary operator " ^
              string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
              string_of_typ t2 ^ " in " ^ string_of_expr e))
        )


        | Init(var, lit) -> let a= type_of_identifier var and b= expr lit in
        (match b with Int -> a
                | _ -> raise(Failure("illegal " ^string_of_typ b^ ", expected int")) )
        | Null(e) -> let k=expr e in (match k with Void -> raise(Failure("no void expression
expected"))
                              |_-> Bool)
        | Ary(var,_) -> let k=function Intptr ->Int
                                     | String_t -> Char
                                     | String_p -> String_t
      in k (type_of_identifier var)
        | Aryasn(var,_,_) ->let k=function Intptr ->Int
                                     | String_t -> Char
                                     | String_p -> String_t
      in k (type_of_identifier var)




    | Unop(op, e) as ex -> let t = expr e in

                (  match op with
                    Neg when t = Int -> Int
                  | Not when t = Bool -> Bool
              | _ -> raise (Failure ("illegal unary operator " ^ string_of_uop op ^
                            string_of_typ t ^ " in " ^ string_of_expr ex))

         )

      | Noexpr -> Void

      | Assign(var, e) as ex -> let lt = type_of_identifier var
                                and rt = expr e in
        check_assign lt rt (Failure ("illegal assignment " ^ string_of_typ lt ^
                                  " = " ^ string_of_typ rt ^ " in " ^
                                  string_of_expr ex))
```

```
    | Call(fname, actuals) as call -> let fd = function_decl fname in
       if fname<>"printx" then

       ( if List.length actuals != List.length fd.formals then
         raise (Failure ("expecting " ^ string_of_int
           (List.length fd.formals) ^ " arguments in " ^ string_of_expr call))
       else
         List.iter2 (fun (ft, _) e -> let et = expr e in
            ignore (check_assign ft et
              (Failure ("illegal actual argument found " ^ string_of_typ et ^
              " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e))))
           fd.formals actuals) else ();
         fd.typ
  in

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

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

    | If(p, b1, b2) -> check_bool_expr p; stmt b1; stmt b2
    | For(e1, e2, e3, st) -> ignore (expr e1); check_bool_expr e2;
                             ignore (expr e3); stmt st
    | While(p, s) -> check_bool_expr p; stmt s
  in

  stmt (Block func.body)

in
List.iter check_function functions
```