BMWSA

(Lack of good abbreviation)

DATA PROCESSING LANGUAGE

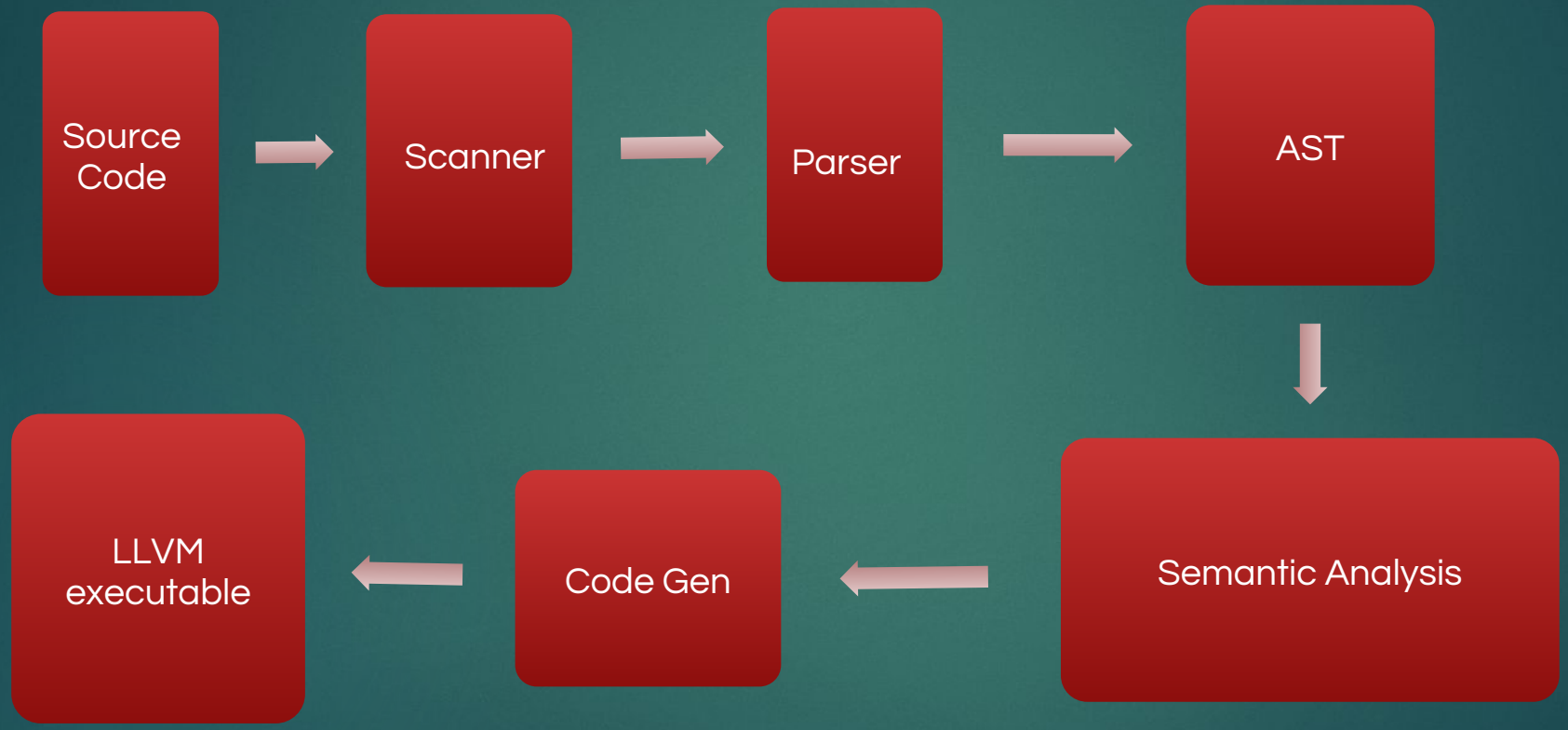
Team Members

- Aman Chahar (ac3946)
  Project Manager, Project Proposal, LRM, Code generation, Test suite
- Miao Yu (my2457)
  Project Proposal, LRM, Code generation, Parser, Scanner, Test suite
- Weiduo Sun (ws2478)
  Project Proposal, LRM
- Sikai Hang (sh3518)
  Project Proposal, LRM
- Baokun Cheng (bc2651)
  Project Proposal, LRM, Library design, Test Suite

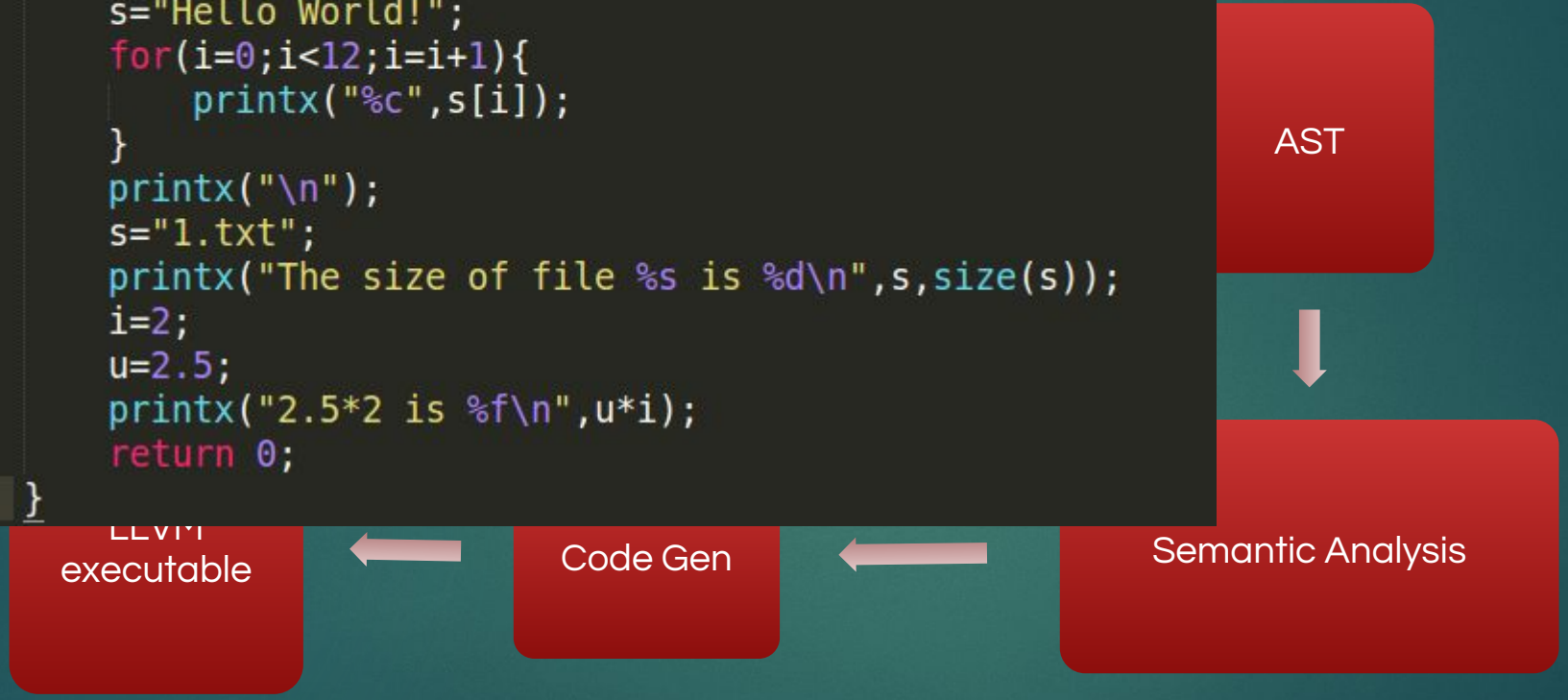
Introduction

- Tremendous amount of data that needs to be processed
- Lot of languages like Python, AWK, R have started with the same goal
- Compiled to LLVM
- Easy split, merge, delete, copy files
- C like syntax
- Library

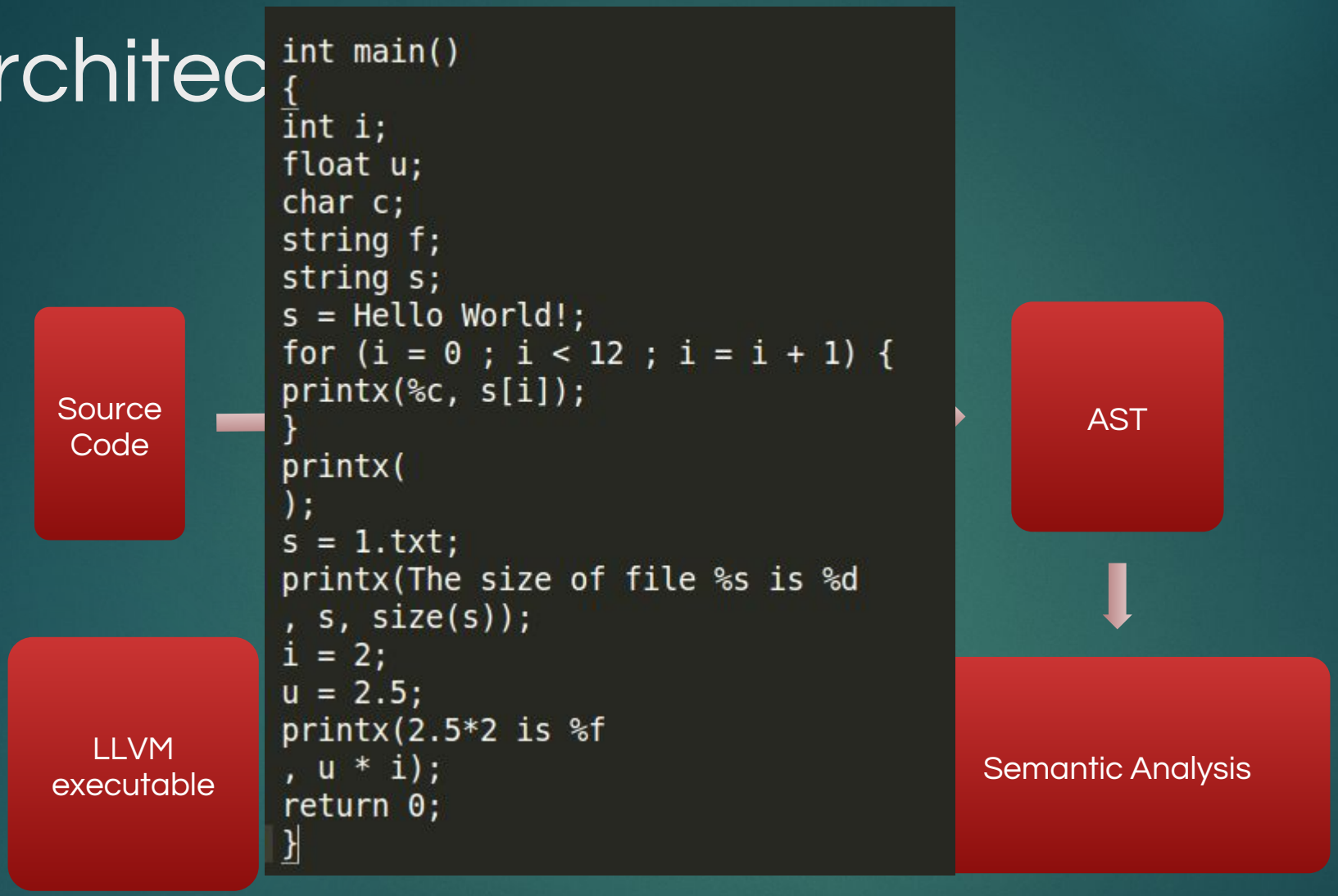
Architecture



```
int main()
{
    int i;
    float u;
    char c;
    file f;
    string s;
    s="Hello World!";
    for(i=0;i<12;i=i+1){
        printx("%c",s[i]);
    }
    printx("\n");
    s="1.txt";
    printx("The size of file %s is %d\n",s,size(s));
    i=2;
    u=2.5;
    printx("2.5*2 is %f\n",u*i);
    return 0;
}
```



Architec


Source Code
LLVM executable
int main()
{
int i;
float u;
char c;
string f;
string s;
s = Hello World!;
for (i = 0 ; i < 12 ; i = i + 1) {
printx(%c, s[i]);
}
printx(
);
s = 1.txt;
printx(The size of file %s is %d
, s, size(s));
i = 2;
u = 2.5;
printx(2.5*2 is %f
, u * i);
return 0;
}
AST
Semantic Analysis

Archit

Source Code

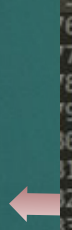
LLVM executo

```
declare i8* @fgets(i8*, i32, i8*)

define i32 @main() {
entry:
  %i = alloca i32
  %u = alloca double
  %c = alloca i8
  %f = alloca i8*
  %s = alloca i8*
  store i8* getelementptr inbounds ([13 x i8]* @tmp1, i32 0, i32 0), i8** %s
  store i32 0, i32* %i
  br label %while

while:                                            ; preds = %while_body, %entry
  %i6 = load i32* %i
  %i7 = load i32* %i
  %i8 = load i32* %i
  %tmp9 = icmp slt i32 %i6, 12
  br i1 %tmp9, label %while_body, label %merge

while_body:                                       ; preds = %while
  %s1 = load i8** %s
  %i2 = load i32* %i
  %tmpp = getelementptr inbounds i8* %s1, i32 %i2
  %deref = load i8* %tmpp
  %tmp1 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([3 x i8]* @tmp11, i32 0, i32 0), i8 %deref)
  %i3 = load i32* %i
  %i4 = load i32* %i
  %i5 = load i32* %i
  %tmp = add i32 %i3, 1
  store i32 %tmp, i32* %i
  br label %while

merge:                                            ; preds = %while
  %tmp110 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([2 x i8]* @tmp12, i32 0, i32 0))
  store i8* getelementptr inbounds ([6 x i8]* @tmp13, i32 0, i32 0), i8** %s
  %s11 = load i8** %s
  %tmp0 = call i8* @fopen(i8* %s11, i8* getelementptr inbounds ([2 x i8]* @fmt4, i32 0, i32 0))
  %tmp7 = call i64 @ftell(i8* %tmp0)
  %s12 = load i8** %s
```

Architecture

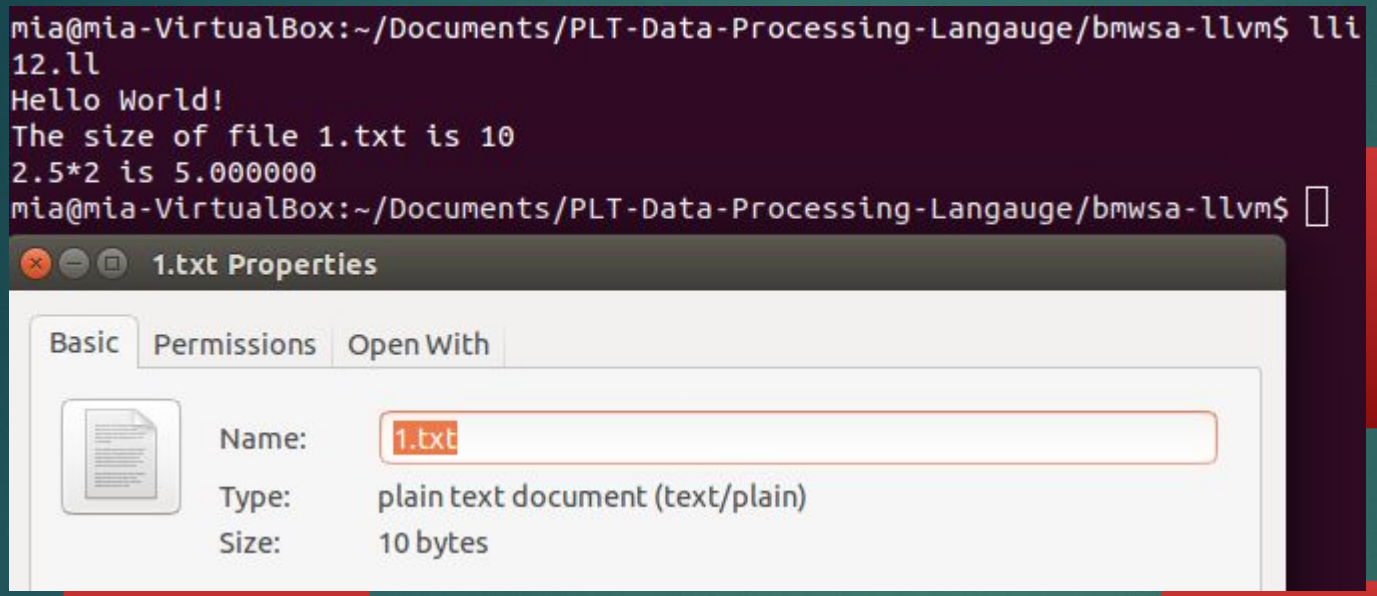
Source Code

LLVM executable



```
define i32 @main() {
entry:
  %i = alloca i32
  %u = alloca double
  %c = alloca i8
  %f = alloca i8*
  %s = alloca i8*
  store i8* getelementptr inbounds ([13 x i8]* @tmp1, i32 0, i32 0), i8** %s
  store i32 0, i32* %i
  br label %while

while:                                            ; preds = %while_body, %entry
  %i6 = load i32* %i
  %i7 = load i32* %i
  %i8 = load i32* %i
  %tmp9 = icmp slt i32 %i6, 12
  br i1 %tmp9, label %while_body, label %merge

while_body:                                       ; preds = %while
  %s1 = load i8** %s
  %i2 = load i32* %i
  %tmpp = getelementptr inbounds i8* %s1, i32 %i2
  %deref = load i8* %tmpp
  %tmp1 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([3 x i8]* @tmp11, i32 0, i32 0), i8 %deref)
  %i3 = load i32* %i
  %i4 = load i32* %i
  %i5 = load i32* %i
  %tmp = add i32 %i3, 1
  store i32 %tmp, i32* %i
  br label %while

merge:                                            ; preds = %while
  %tmp110 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([2 x i8]* @tmp12, i32 0, i32 0))
  store i8* getelementptr inbounds ([6 x i8]* @tmp13, i32 0, i32 0), i8** %s
  %s11 = load i8** %s
  %tmp0 = call i8* @fopen(i8* %s11, i8* getelementptr inbounds ([2 x i8]* @fmt4, i32 0, i32 0))
  %tmp7 = call i64 @ftell(i8* %tmp0)
  %s12 = load i8** %s
  %tmp113 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([27 x i8]* @tmp14, i32 0, i32 0), i8* %s12, i64 %tmp7)
  store i32 2, i32* %i
  store double 2.500000e+00, double* %u
```

Architecture



AST

Semantic Analysis

Code Gen

LLVM executable

Parser

```
%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA RBRACKET LBRACKET INCLUDE
%token PLUS MINUS TIMES DIVIDE ASSIGN NOT SPLUS SMINUS
%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:
    inc_libs decls EOF { Program($1,$2) }

inc_libs:
    /* nothing */ { [] }
    |   inc_lib_list  { List.rev $1 }

inc_lib_list:
      inc_lib_decl              { [$1] }
    |   inc_lib_list inc_lib_decl { $2::$1 }

inc_lib_decl:
  INCLUDE LPAREN STRING_LITERAL RPAREN SEMI { inc_lib($3) }


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

```
typ:
    INT { Int }
  | BOOL { Bool }
  | VOID { Void }
  | FLOAT { Float }
  | CHAR { Char }
  | STRING {String_t}
  | INT TIMES { Intptr}
  | STRING TIMES { String_p }


array_t:
  typ ID LBRACKET brackets RBRACKET {    L($1,$2,Arraytype($1,$4)) }

dtype:
   typ {  Dtype($1)  }



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) }
```

AST and Pretty printing functions

```
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 -> ""

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"

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
```

```
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
  | L of typ * string * dtype
  | Ary of string* expr
  | Aryasn of string* expr* expr
  | Vectors of typ*string * 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
```

Code Gen

```
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_int (ltype_of_typ t) 0
      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|])
```

```
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.const_intcast sz (i32_t) false 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.Binop (e1, op, e2) ->

        let e1' = expr builder e1
        and e2' = expr builder e2 in
        let tp1=(L.type_of (L.const_int i32_t 3)) and tp2=(L.type_of (L.const_float f_t 3.2))
          and tp3=(L.type_of e1') in

        (match op with

    A.Add     -> (if tp1=tp2 then (L.build_fadd) else (L.build_add))
  | A.Sub     -> (if tp1=tp2 then (L.build_fsub) else (L.build_sub))
```

Language syntax

Data types

- Int
- Boolean
- Float
- Char
- File
- Arrays (String, Int, String array)

Library Functions

- Open file
- Close File
- Count lines in a file
- Split a file by a line number
- Merge file
- Delete a file
- Print
- Split String
- …

Sample codes

Hex characters, type casting

```
int main()
{
    int p, float u, char p;

    k='c';

    if(k>'b')
        printx("%c\n",k);

    printx("%c\n",k+'\x08');

    u=4.0;
    p=2;

    printx("%f\n",u+p);

    return 0;
}
```

Merge file

```
void mergefile(string object, string path1, string path2){

    copyfile(path1,object);

    fputs("\n",fopen(object,"a"));

    copyfile(path2,object);

}
```

Split string, String array

```
int main()
{
    string a;
    string *d;
    int i;
    a="Aman Chahar Miao Yu Baokun Cheng Sikai Huang this is a sample code sepearted
    d=splitstring(a,30," ");

    for(i=0;i<30;i=i+1){
        printx("%s\n",d[i]);
    }

    return 0;
}
```

Some more library functions

string itos (*int* a) —> convert int to string

bool match(*string* s, *char* a) —> return true if a is in the string, otherwise false

bool strcmp(*string* s1, *string* s2) —> return true if two string have same content

void deleteword(*string* filepath, *string* word)—>delete the word in a file, returns the count of the word

void replacewords(*string* filepath, *string* word, *string* replace) —> replace the word with 'replace' and return the count of the word

int searchwords(*string* path, *string* word)—> returns the count of the word

void insert (*string* path, *string* content, *int* ln, *int* col) —> insert content into the specific position denoted by line and column, warns failure if ln or col exceeds the boundary

char getChar(*string* path, *int* ln, *int* col) —> get the char at specific position, return same as insert if out of boundary

Some more library functions

int getLine(*string* path, *int* ln) —> print the line with line number ln, returns 1 if succeed, and returns 0 if fail

void deleteLine(*string* path, *int* start, *int* end) —> delete lines between line number start and end in given file

void countLine(string path, *int* ln) —> delete the line with line number ln

void splitfile(*string* path1, *string* path2, *string* original, *int* ln, *int* col) —> split the original file into two separate files with path1 and path2, from the specific position

void mergefile(*string* result, *string* path1, *string* path2) —> merger two files in path1 and path2 into one file, with path result

void copyfile(*string* result, *string* original)—> copy the original file to the result path

Test Suite

- Designed around 100 tests
- Tested for both correct and incorrect syntax
- Automated test script to evaluate all the test cases

```
fail-assign3.bmwsa          fail-global1.bmwsa          test-float1.out             test-if1.out
fail-assign3.err            fail-global1.err            test-floatintadd.bmwsa      test-if2.bmwsa
fail-dead1.bmwsa            fail-global2.bmwsa          test-floatintadd.out        test-if2.out
fail-dead1.err              fail-global2.err            test-fopen.bmwsa            test-if3.bmwsa
fail-dead2.bmwsa            fail-if1.bmwsa              test-fopen.out              test-if3.out
fail-dead2.err              fail-if1.err                test-for1.bmwsa             test-if4.bmwsa
fail-expr1.bmwsa            fail-if2.bmwsa              test-for1.out               test-if4.out
fail-expr1.err              fail-if2.err                test-for2.bmwsa             test-if5.bmwsa
fail-expr2.bmwsa            fail-if3.bmwsa              test-for2.out               test-if5.out
fail-expr2.err              fail-if3.err                test-func1.bmwsa            test-include.bmwsa
fail-floatassign1.bmwsa     fail-nomain.bmwsa           test-func1.out              test-includedeclare.bmwsa
fail-floatassign1.err       fail-nomain.err             test-func2.bmwsa            test-includedeclare.out
fail-for1.bmwsa             fail-return1.bmwsa          test-func2.out              test-includefunction.bmwsa
fail-for1.err               fail-return1.err            test-func3.bmwsa            test-includefunction.out
fail-for2.bmwsa             fail-return2.bmwsa          test-func3.out              test-include.out
fail-for2.err               fail-return2.err            test-func4.bmwsa            test-local1.bmwsa
fail-for3.bmwsa             fail-while1.bmwsa           test-func4.out              test-local1.out
fail-for3.err               fail-while1.err             test-func5.bmwsa            test-local2.bmwsa
fail-for4.bmwsa             fail-while2.bmwsa           test-func5.out              test-local2.out
fail-for4.err               fail-while2.err             test-func6.bmwsa            test-ops1.bmwsa
fail-for5.bmwsa             fopen.txt                   test-func6.out              test-ops1.out
fail-for5.err               test-add1.bmwsa             test-func7.bmwsa            test-ops2.bmwsa
fail-func1.bmwsa            test-add1.out               test-func7.out              test-ops2.out
fail-func1.err              test-arith1.bmwsa           test-func8.bmwsa            test-stringconcat_print.bmwsa
fail-func2.bmwsa            test-arith1.out             test-func8.out              test-stringconcat_print.out
fail-func2.err              test-arith2.bmwsa           test-gcd2.bmwsa             test-var1.bmwsa
fail-func3.bmwsa            test-arith2.out             test-gcd2.out               test-var1.out
fail-func3.err              test-arith3.bmwsa           test-gcd.bmwsa              test-var2.bmwsa
fail-func4.bmwsa            test-arith3.out             test-gcd.out                test-var2.out
fail-func4.err              test-comments2.bmwsa        test-global1.bmwsa          test-while1.bmwsa
fail-func5.bmwsa            test-comments2.out          test-global1.out            test-while1.out
fail-func5.err              test-comments.bmwsa         test-global2.bmwsa          test-while2.bmwsa
fail-func6.bmwsa            test-comments.out           test-global2.out            test-while2.out
fail-func6.err              test-countlines.bmwsa       test-global3.bmwsa
fail-func7.bmwsa            test-countlines.out         test-global3.out
fail-func7.err              test-fib.bmwsa              test-hello.bmwsa
```

Development and Challenges

- Version control (and merge challenges)
- Weekly meetings
- Julie (TA) giving constant feedback and guidance
- LLVM!
  - Defining basic Datatypes like String and Arrays are also challenging
  - Steep learning curve!
- Shift/Reduce and Reduce/Reduce conflicts

Demo Code

- We decided to choose some unformatted files
- Used to evaluate data processing tools at Columbia CSDS course
- Used python and awk/sed/grep to get same results as our language

HTML Files

- Worldcup
- 2013films