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

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

Source Code

LLVM executable
Architecture

LLVM executable

Code Gen

Semantic Analysis

AST

LLVM

Hello World!
The size of file 1.txt is 10
2.5*2 is 5.000000

miamia-VirtualBox:/Documents/PLT-Data-Processing-Language/bmwsa-llvm$ lli
12.ll
1

miamia-VirtualBox:/Documents/PLT-Data-Processing-Language/bmwsa-llvm$
Parser

```plaintext
#token SEMI LPAREN RPAREN LBRACE RBRACE COMMA RBRACKET LBRACKET INCLUDE
#token PLUS MINUS TIMES DIVIDE ASSIGN NOT PLUS MINUS
#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 <char> CHAR_LITERAL
#token EOF

nonassoc NOELSE
nonassoc ELSE
right ASSIGN
left OR
left AND
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 }
  INT TIMES { Int ptr }
  STRING TIMES { String ptr }

array_t:
  typ ID LBRACKET brackets RBRACKET { L($1, $2, Arraytype($3, $4)) }

dtype:
  Typ { Dtype($1) }

brackets:
  { } 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 NOELSE { If($3, $5, Block([$1])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt { If($3, $5, [$7]) }
  | FOR LPAREN expr opt SEMI expr opt SEMI expr opt RPAREN stmt { For($3, $5, $7, $9) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5) }
```

let rec string_of_expr = function
  | Literal(l) -> string_of_int l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | StringLit(s) -> """" s """"
  | If(s, e1, e2) -> string_of_expr e1 " if " string_of_expr e2
  | Binop(e, o, e2) -> """" string_of_expr e1 o """" string_of_expr e2
  | Unop(o, e) -> string_of_unop o " string_of_expr e
  | Assign(v, e) -> v " = " string_of_expr e
  | Call(f, e, ...) -> String.concat "( " (List.map string_of_expr e) ")"
  | Noexpr -> ""

let rec string_of_stmt = function
  | Block(steps) ->
    "\n\n" String.concat "\n" (List.map string_of_stmt steps) "\n\n"
  | Expr(expr) -> string_of_expr expr ";" "\n"
  | Return(expr) -> "return" " string_of_expr expr ";" "\n"
  | If(e, s1, s2) -> "if " string_of_expr e " if " string_of_stmt s1 " else " string_of_stmt s2
  | For(e, s1, s2, s3) ->
    "for " string_of_expr e ";" " string_of_expr s1 ";" " string_of_expr s2 ";" " string_of_stmt s3 "") " string_of_stmt s1 ";" " string_of_expr e "= " string_of_expr e "=" " string_of_stmt s1
  | While(e, s1) -> "while " string_of_expr e " do " string_of_stmt s1 ";" " string_of_expr e "=" " string_of_expr e "=" " string_of_stmt s1

let string_of_typ = function
  | Int -> "int"
  | Bool -> "bool"
  | Void => "void"
  | String_t => "string"
  | Float t => "float"
  | Char _ => "char"

let string_of_vdecl (t, id) = string_of_typ t " " id " = " "

let string_of_fdecl fdecl =
  string_of_typ fdecl.ty " "
  fdecl.name "=" "( " String.concat "," (List.map snd fdecl.formals) "") "\n\n"
  String.concat "( " (List.map string_of_vdecl fdecl.locals) " \n\n"
  String.concat "( " (List.map string_of_stmt fdecl.body) " \n\n"

let string_of_program (Program(first, second)) =
  let (vars, funcs) = second in
  let type program = Program of include_stmt list * decls_val
      type include_stmt = Include of string
      type decls_val = bind list * func_decl list
      type func_decl = {
        typ : typ;
        name : string;
        formals : bind list;
        locals : bind list;
        body : stmt list;
      }
      type program = Program of include_stmt list * decls_val

let type uop = Neg | Not
  | typ = Int | Void | Char | Float | String_t | IntPtr | StringType | Dtype of typ

let typ bind = typ * string
  | expr =
    | Literal of int
    | Float_lit of float
    | String_lit of string
    | Char_lit of char
    | BoolLit of bool
    | If(d, e1, e2) -> string_of_expr e1 " if " string_of_expr e2
    | Binop(e, o, e2) -> "" string_of_expr e1 o "" string_of_expr e2
    | Unop(o, e) -> string_of_unop o " string_of_expr e
    | Assign(v, e) -> v " = " string_of_expr e
    | Call(f, e, ...) -> String.concat "( " (List.map string_of_expr e) ")"
    | Noexpr -> ""

let type string =
  | 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
  | Include of string

let type decls_val = bind list * func_decl list
  | type decls_val = bind list * func_decl list
  | type decls_val = bind list * func_decl list
  | type decls_val = bind list * func_decl list
  | type decls_val = bind list * func_decl list
  | type decls_val = bind list * func_decl list
  | type decls_val = bind list * func_decl list
  | type decls_val = bind list * func_decl list
  | type decls_val = bind list * func_decl list
  | type decls_val = bind list * func_decl list
  | type decls_val = bind list * func_decl list
  | type decls_val = bind list * func_decl list
  | type decls_val = bind list * func_decl list
  | type decls_val = bind list * func_de
let rec translate (A.Program(first, second), A.Program(first1,second1)) =
(* Translate a program A to a program B *)
let (globals, functions) = second
and (globals2, functions2) = second1 in
let functions = List.append functions2 functions in
let context = L.global_context () in
let module = L.create_module context "Omvsa"

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 if_t = L.I1_type context
and str_t = L.stype context
(* and typ_t = L.typ_context()) =
let str_t = L.pointer_type I8_t
and void_str = L.pointer_type I32_t in

let type_of_typ = function
| L.Int t -> I64_t
| L.Bool -> I8_t
| L.Void -> void_str
| L.Float f -> f
| L.Char c -> I8_t
| L.String str -> str_t
| L.Intptr t -> str_t
| L.Arr t -> L.pointer_type I32_t
| L.Arr_str p -> L.pointer_type I8_t
(* declare each globa variable; remember its value in a map *)

and global_vars =
let global_vars = (t, n) =
let init L.const_int (type of t) a =
in StringMap.add n ((define global n init the module) a) in
List.fold_left global_vars StringMap.empty globals in

let rec expr builder = function
| A.Literal i -> L.const_int 12_d t i
| A.Quot s -> L.const_int 12_d t i
| 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 t c
| A.Noexpr -> L.const_int 12_d 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 boundaries at para1 [] [para2] "tmp" builder in
L.build_load k "derefer" builder
| A_arys (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 boundaries at para1 [] [para2] "tmp" builder in
L.build_store para3 k builder
| A.Init (e1, e2) -> let cnt = (lookup e1) and cnt2 = expr builder e2 in

| A.Binop (e1, op, e2) ->

let i1 = expr builder e1

and i2 = expr builder e2 in
let t1 = (let t = (L.type_of (L.const_int I32_t 0)) and t2 = (L.type_of (L.const_float F_t 0)))

and t3 = (L.type_of (L.Id e1)) in

(match op with
| A.Add -> if t1 = t2 then (L.build_add) else (L.build_add)
| A.Sub -> if t1 = t2 then (L.build_sub) 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

```c
int main()
{
    int p, float u, char p;
    k='c';
    if(k>'b')
        printf("%c\n",k);
    printf("%c\n",k+'\x08');
    u=4.0;
    p=2;
    printf("%f\n",u+p);
    return 0;
}
```

Merge file

```c
void mergefile(string object, string path1, string path2){
    copyfile(path1,object);
    fputs("\n",fopen(object,"a"));
    copyfile(path2,object);
}
```

Split string, String array

```c
int main()
{
    string a;
    string *d;
    int i;
    a="Aman Chahar Miao Yu Baokun Cheng Sikai Huang this is a sample code sepearted"
    d=splithstring(a,30,"");
    for(i=0;i<30;i=i+1){
        printf("%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
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
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