SER502-Spring2017-Team10

Guo Xiangyu
Zhu Rundong
Feng Ximing
MacArthur Katie

Outline

- Language Name
- Grammar Rule
- Byte Code
- Compiler
- Runtime
- Highlights
- Retrospects
- Future works

Language name

- **TEN** is our language name since we are team **TEN**.
- Also, in Chinese “ten” means perfect, we made our small language project as perfect as we can in the limited time.

Grammar of our language

```
grammar ten;
program : stmt_list ;
stmt_list : stmt ';' stmt_list | stmt ';' ;
stmt : decl_stmt | assign_stmt | if_stmt | for_stmt | print_stmt | ;
decl_stmt : 'var' ID ;
assign_stmt : ID 'is' expr ;
if_stmt : 'if' '(' boolean_expr ')' 'then' '{' stmt_list '}' |
          'if' '(' boolean_expr ')' 'then' '{' stmt_list '}' 'else' '{' stmt_list '}' ;
for_stmt : 'for' ID 'from' expr 'to' expr 'step' expr '{' stmt_list '}' | 'for' ID 'from'
expr 'downto' expr 'step' expr '{' stmt_list '}' ;
print_stmt : 'print' expr ;
boolean_expr : expr '=' expr | expr '<>' expr |
               expr '<' expr | expr '<=' expr |
               expr '>' expr | expr '>=' expr |
               boolean_val ;
boolean_val : 'true' | 'false' ;
expr : term res1;
res1 : '+' term res1 | '-' term res1 | ;
term : factor res2;
res2 : '*' factor res2 | '/' factor res2 | '%' factor res2 |  ;
factor : '(' expr ')' | NUMBER | ID ;
NUMBER : [-]?[0-9]+ ;
ID : [a-z|A-Z]+ ;
WS : [ \t\r\n]+ -> skip ;
```

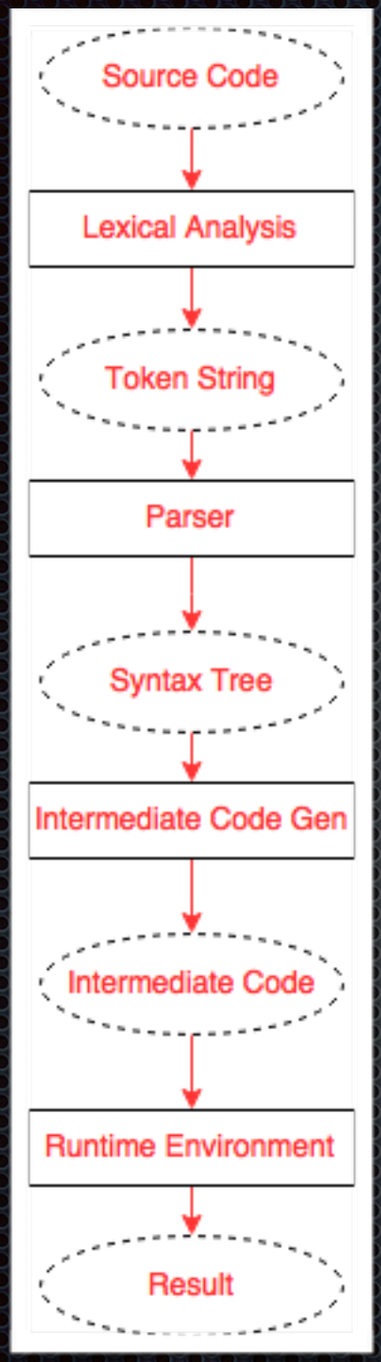
- We supported boolean expression as well as integer type.
- We are using “is” to do the assignment.
- We are using “var” to declare variable.
- We supported “if-then” and “if-then-else” statement to make decision.
- We supported “for” statement for iterative execution.
- In addition, we supported “print” statement to print out the value or variables in our virtual machine.

Byte code of our runtime

```
Type 1: declare
DEC var1                        ; declare a variable

Type 2: assignment
MOV var1, var2/value            ; assign var2/value to var1

Type 3: arithmetic operation
ADD var1, var2/value            ; add var2/value to var1
SUB var1, var2/value            ; sub var2/value from var1
MUL var1, var2/value            ; mul var2/value to var1
DIV var1, var2/value            ; div var2/value from var1
MOD var1, var2/value            ; mod var1 with var2/value

Type 4: compare and jump operation
CMP var1/value, var2/value      ; CMP var1/value and var2/value, set $flag = var1 - var2/
value.
JE  label                       ; Jump to label if $flag Equal to zero.
JNE label                       ; Jump to label if $flag Not Equal to zero.
JL  label                       ; Jump to label if $flag Less than zero.
JLE label                       ; Jump to label if $flag Less than or Equal to zero.
JG  label                       ; Jump to label if $flag Great than zero.
JGE label                       ; Jump to label if $flag Great than or Equal to zero.
JMP label                       ; Jump to label, always.

Type 5: output result
OUT var1/value                  ; Output var1/value.
```

- Type 1: declare a variable using “DEC”
- Type 2: assign variable2/value to variable1 using “MOV”
- Type 3: perform arithmetic operation between variable1 and variable2.
- Type 4: compare var1/value with var2/value, and jump based on the comparison result.
- Type 5: output the variable inside the machine storage or instant value.

Compiler - Processing



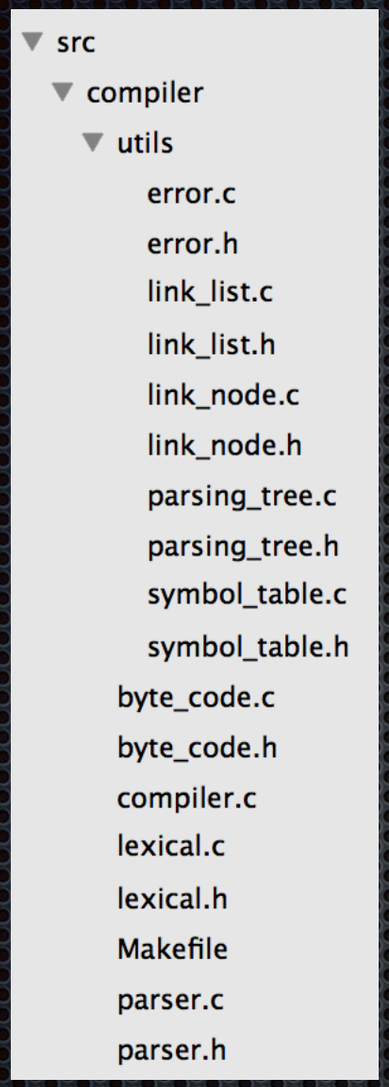
- Lexical Analysis:
  Input: source code.
  Processing: tokenize, construct symbol table.
  Output: token string, symbol table.
- Parser:
  Input: token string, symbol table.
  Processing: construct parsing tree.
  Output: parsing tree.
- Bytecode:
  Input: parsing tree.
  Processing: generate byte code.
  Output: byte code.

Compiler - Architecture

```
symbol_table_st *symbol_table = symbol_table_init();
if (symbol_table == NULL)
    return ENOMEM;

link_list_st *token_list = lexical_analysis(symbol_table);
if (token_list == NULL)
    return ENOMEM;

parsing_tree_st *parse_tree = syntax_analysis(token_list, symbol_table);
if (parse_tree == NULL)
    return ENOMEM;

link_list_free(token_list);

link_list_st *byte_code = semantic_analysis(parse_tree);
if (byte_code == NULL)
    return ENOMEM;

parsing_tree_free(parse_tree);

link_list_traverse(byte_code, print_byte_code, NULL);
```

- Convert processing into interface of each component.

Compiler - Architecture



- Folder “utils” for basic data structure.
- lexical.* for lexical analysis component.
- parser.* for syntax analysis component.
- byte_code.* for semantic analysis component.
- compiler.c for glue all components together.

Compiler - Data Structures

- Symbol Table
  Used for store token and the token type.
- Linked List
  Used in token list, byte code list.
- Binary Tree in Left Child Right Sibling Format.
  Used in parsing tree.

Compiler - Data Structures

```c
#ifdef PARSING_TREE_TEST

int test_print_cb(parsing_tree_st *tree_node, void *cb_data) {
}

void test_suite_three() {
}

void test_suite_two() {
}

void test_suite_one() {
}

int main() {
}

#endif
```

```c
#ifdef SYMBOL_TABLE_TEST

void test_case_one() {
}

void test_case_two() {
}

void test_case_three() {
}

int main() {
}

#endif
```

```c
#ifdef LINK_NODE_TEST

void test_suite_three() {
}

void test_suite_two() {
}

void test_suite_one() {
}

int main() {
}

#endif
```

```c
#ifdef LINK_LIST_TEST

int test_print(link_node_st *node, void *data) {
}

void test_suite_three() {
}

void test_suite_two() {
}

void test_suite_one() {
}

int main() {
}

#endif
```

- Unit test cases inside the basic data structure, to make sure the reliability of basic data structures.

Compiler - Lexical Analysis

```
while ((char_in = getchar()) != EOF) {
    if (char_in == ' ' || char_in == '\t' || ...

    if (char_in == '(' || char_in == ')' || ...
    } else if (isalpha(char_in)) {...
    } else if (char_in == '=' || char_in == '<' || char_in == '>') {...
    } else if (char_in == '+' || char_in == '%' || ...
    } else if (char_in == '-') {...
    } else if (isdigit(char_in)) {...
    }
}
```

- First, the symbol table is initialized, containing all of the keywords from the grammar rule, parentheses, brackets, and the delimiter
- Next, the lexical analyzer takes in the code from the source code file and tokenizes it store in a linked list structure
- Everything in the code is also looked up in the symbol table. Whatever is not found in the symbol table is added in.

Compiler - Parser

```
static parsing_tree_st *generate_decl_stmt(link_list_st *, symbol_table_st *);
static parsing_tree_st *generate_print_stmt(link_list_st *, symbol_table_st *);
static parsing_tree_st *generate_assign_stmt(link_list_st *, symbol_table_st *);
static parsing_tree_st *generate_if_stmt(link_list_st *, symbol_table_st *);
static parsing_tree_st *generate_for_stmt(link_list_st *, symbol_table_st *);
static parsing_tree_st *generate_expre(link_list_st *, symbol_table_st *);
static parsing_tree_st *generate_term(link_list_st *, symbol_table_st *);
static parsing_tree_st *generate_factor(link_list_st *, symbol_table_st *);
static parsing_tree_st *generate_res1(link_list_st *, symbol_table_st *);
static parsing_tree_st *generate_res2(link_list_st *, symbol_table_st *);
static parsing_tree_st *generate_stmt(link_list_st *, symbol_table_st *);
static parsing_tree_st *generate_boolean_expre(link_list_st *, symbol_table_st *
static parsing_tree_st *generate_boolean_value(link_list_st *, symbol_table_st *
static parsing_tree_st *generate_stmt_list(link_list_st *, symbol_table_st *);
```

- First, take token list and and symbol as parameters.
- Pop each element at one time and verify the token. If it matched the grammar rule, generate the node on the parsing tree according to the grammar rules in a top-down way.
- Once the whole process is done, all tokens has been consumed, it would generate a complete parsing tree.

Compiler - Code Generator

```
static int handle_if_stmt_helper(parsing_tree_st *then_node);
static void byte_code_new(link_list_st *, char *, char *, char *);
static void handle_stmt_list(parsing_tree_st *, link_list_st *);
static void handle_stmt(parsing_tree_st *, link_list_st *);
static char *handle_boolean_value(parsing_tree_st *, link_list_st *);
static char *handle_boolean_expr(parsing_tree_st *, link_list_st *);
static void handle_if_stmt(parsing_tree_st *, link_list_st *);
static void handle_for_stmt(parsing_tree_st *, link_list_st *);
static char *handle_expr(parsing_tree_st *, link_list_st *);
static char *handle_term(parsing_tree_st *, link_list_st *);
static void handle_decl_stmt(parsing_tree_st *, link_list_st *);
static void handle_assign_stmt(parsing_tree_st *, link_list_st *);
static void handle_print_stmt(parsing_tree_st *, link_list_st *);
static char *handle_res1(parsing_tree_st *, link_list_st *, char *);
static char *handle_factor(parsing_tree_st *, link_list_st *);
static char *handle_res2(parsing_tree_st *, link_list_st *, char *);
```

- First, get the entrance of parsing tree node and data in tree node.
- Then, according to parsing tree structure traverse each tree nodes value check tree node satisfied with syntax rule or not.
- Get value of each node and figure out byte_code append to link list accordingly .

Compiler - Component Test

```
#ifdef XTEST

int print_list(link_node_st *node, void *cb_data)
}

void test_case_one(char *file_name) {
}

void test_case_two(char *file_name) {
}


int main(int argc, char *argv[]) {
}
#endif
```

```
#ifdef XTEST

/**
int print_data(parsing_tree
}

int print_list_data(link_no
}

/**
void test_case_one() {
}

void test_case_two() {
}

/**
void test_case_three() {
}

/**
void test_case_four() {

}

void test_case_five() {
}

int main()
{
}

#endif
```

```
#ifdef XTEST
//tree structure

parsing_tree_st *setup_expr_tr
}

int print_byte_code(link_node_
}

int print_tree(parsing_tree_st
}

void test_suite_one() {
}

void test_suite_two() {
}

void test_suite_three() {
}

void test_suite_four() {
}

void test_suite_five() {
}

void test_suite_six() {
}

void test_suite_seven() {
}

int main() {
}

#endif // XTEST
```

- Unit test on each component to make sure they worked properly before integrated together.

Compiler - Snapshots

```
GuoXiangYusMacBook-Pro:compiler tobielf$ make
[Clean]
[CC] utils/error.c
[CC] utils/link_node.c
[CC] utils/link_list.c
[CC] utils/symbol_table.c
[CC] utils/parsing_tree.c
[CC] lexical.c
[CC] parser.c
[CC] byte_code.c
[CC] compiler.c
[linking compiler]
GuoXiangYusMacBook-Pro:compiler tobielf$ ls
Makefile    byte_code.o compiler    demo.asm
byte_code.c code.asm    compiler.c  demo.ten
byte_code.h code.ten    compiler.o  lexical.c
```

- Then we combine all components together.
- Successfully built the compiler without any warning message.

Compiler - Snapshots

```
GuoXiangYusMacBook-Pro:compiler tobielf$ cat demo.ten
var i;
i is -1 + 2 * 3;
print i;
GuoXiangYusMacBook-Pro:compiler tobielf$ ./compiler demo.ten demo.asm
GuoXiangYusMacBook-Pro:compiler tobielf$ cat demo.asm
DEC i
DEC _temp2
MOV _temp2 2
MUL _temp2 3
DEC _temp1
MOV _temp1 -1
ADD _temp1 _temp2
MOV i _temp1
OUT i
```

- Translated our high-level language into our byte code.

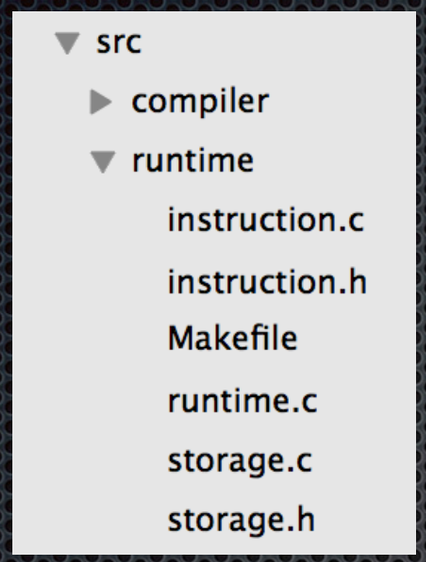
Runtime - Architecture

```c
int main(int argc, char *argv[])
{
    struct stat file_stat;

    if (argc != 2) {...
    }

    if (stat(argv[1], &file_stat) != 0) {...
    }

    s_machine_store = machine_memory_init();

    s_instructions = instruction_load_program(argv[1]);

    s_evaluate(s_instructions);

    instruction_clean_up(s_instructions);

    machine_memory_fini(s_machine_store);

    return 0;
}
```

```c
operation_st g_operations[] = { {"DEC", eval_dec},
                                {"MOV", eval_mov},
                                {"OUT", eval_out},
                                {"ADD", eval_add},
                                {"SUB", eval_sub},
                                {"MUL", eval_mul},
                                {"DIV", eval_div},
                                {"MOD", eval_mod},
                                {"CMP", eval_cmp},
                                {"JE" , eval_je },
                                {"JNE", eval_jne},
                                {"JL" , eval_jl },
                                {"JLE", eval_jle},
                                {"JG" , eval_jg },
                                {"JGE", eval_jge},
                                {"JMP", eval_jmp},
                                {NULL , NULL} };
```

- Initialize the machine memory storage.
- Load byte code into runtime.
- Do the evaluation instruction by instruction, using the corresponding eval_* function.

Runtime - Architecture



- instruction.* for loading and analyzing byte code in to the runtime
- storage.* for runtime storage management
- runtime.c for all evaluate function and the evaluate function will perform operation on runtime storage based on the instruction.

Runtime - Instruction Set

```c
typedef struct instruction instruction_st;
struct instruction;

typedef struct instruction_set instruction_set_st;
struct instruction_set;

/**...
instruction_set_st *instruction_load_program(const char *);

/**...
void instruction_clean_up(instruction_set_st *);

/**...
instruction_st *instruction_set_get_instruction(instruction_set_st *);

/**...
void instruction_set_set_pc(instruction_set_st *, int);
```

- Load byte code form an instruction set.
- Fetch instruction based on the Program Counter.
- Set the Program Counter to a new address

Runtime - Machine Storage

```c
typedef struct memory memory_st;
struct memory;

typedef struct machine_memory machine_memory_st;
struct machine_memory;

/**...
machine_memory_st *machine_memory_init();

/**...
void machine_memory_fini(machine_memory_st *);

/**...
memory_st* machine_memory_get_variable(machine_memory_st *, char *, int);

/**...
int machine_memory_set_variable(machine_memory_st *, char *, int, int);

/**...
void machine_memory_open_scope(machine_memory_st *);

/**...
void machine_memory_close_scope(machine_memory_st *);
```

- Get/put variable from/to the machine storage.
- Open/close the variable scope.

Runtime - Evaluate

```c
/**
static void eval_bin_op_helper(void *, void *, char);

/**
static void eval_dec(void *, void *);

/**
static void eval_mov(void *, void *);

/**
static void eval_out(void *, void *);

/**
static void eval_add(void *, void *);

/**
static void eval_sub(void *, void *);

/**
static void eval_mul(void *, void *);

/**
static void eval_div(void *, void *);

/**
static void eval_mod(void *, void *);
```

```c
/**
static void eval_cmp(void *, void *);

/**
static void eval_je(void *, void *);

/**
static void eval_jne(void *, void *);

/**
static void eval_jl(void *, void *);

/**
static void eval_jle(void *, void *);

/**
static void eval_jg(void *, void *);

/**
static void eval_jge(void *, void *);

/**
static void eval_jmp(void *, void *);

/**
static void eval_bool_op_helper(void *, void *, cmp_cb flag_cmp);
```

- Evaluate the byte code based on the operation code.

Runtime - Variable Scope

```
        // find method for op_code
        i = 0;
        while (g_operations[i].op_code != NULL) {...
        }

        if (g_operations[i].op_code == NULL) {
#ifdef DEBUG
            fprintf(stderr, "label change scope\n");
#endif
            // label change scope.(for1: scope++, for1_end: scope--)
            if (strstr(op_code, "_end:"))
                machine_memory_close_scope(s_machine_store);
            else
                machine_memory_open_scope(s_machine_store);
        }
```

- When it comes to the label in byte code, no corresponding eval operations will performed.
- So we handled the variable scope based on label generated by “for” or “if” statement.
- For labels like “for#:” indicated opening a new scope, for labels like “for#_end:” indicated exiting a scope, so the runtime will release all the variables on that scope.

Runtime - Variable Scope

```c
/**
 * @brief evaluate function of "DEC" instruction
 * @param first first operand.
 * @param second second operand.
 */
static void eval_dec(void *first_operand, void *second_operand) {
    memory_st *variable_memory;
    char *variable_name = NULL;

    if (first_operand == NULL)
        return;

    variable_name = (char *)first_operand;
    // check on this scope, if exist warning redefine and exit.
    variable_memory = machine_memory_get_variable(s_machine_store,
                                                  variable_name,
                                                  MEMORY_CURRENT_SCOPE);
```

```c
/**
 * @brief evaluate function of "OUT" instruction
 * @param first first operand.
 * @param second second operand.
 */
static void eval_out(void *first_operand, void *second_operand) {
    memory_st *variable_memory;
    char *var_one = NULL;
    int value = 0;

    if (first_operand == NULL)
        return;

    var_one = (char *)first_operand;

    if (isalpha(var_one[0]) || var_one[0] == '_') {
        // check var_one on ALL scope, if doesn't exist, warning and exit.
        variable_memory = machine_memory_get_variable(s_machine_store,
                                                      var_one,
                                                      MEMORY_ALL_SCOPE);
```

- For declare a new variable(left side), run time will only search on current scope, check the variable exist or not, if exist that is a redefine warning.
- For reference an old variable(right side), run time will search on the whole scope, from most recent to least recent order.

Runtime - Snapshot

```
GuoXiangYusMacBook-Pro:runtime tobielf$ make
[Clean]
[CC] runtime.c
[CC] instruction.c
[CC] storage.c
[linking runtime]
GuoXiangYusMacBook-Pro:runtime tobielf$ ls
Makefile      instruction.c result.txt    run
debug.txt     instruction.h runtime       sto
demo.asm      instruction.o runtime.c     sto
```

- Successfully built the runtime without any warning message.

- Execute the byte code and get the result

```
GuoXiangYusMacBook-Pro:runtime tobielf$ cat demo.asm
DEC i
DEC _temp2
MOV _temp2 2
MUL _temp2 3
DEC _temp1
MOV _temp1 -1
ADD _temp1 _temp2
MOV i _temp1
OUT i
GuoXiangYusMacBook-Pro:runtime tobielf$ ./runtime demo.asm
5
```

Highlights - Agile method

- We divided two weeks into four developing phases, two phases per week.

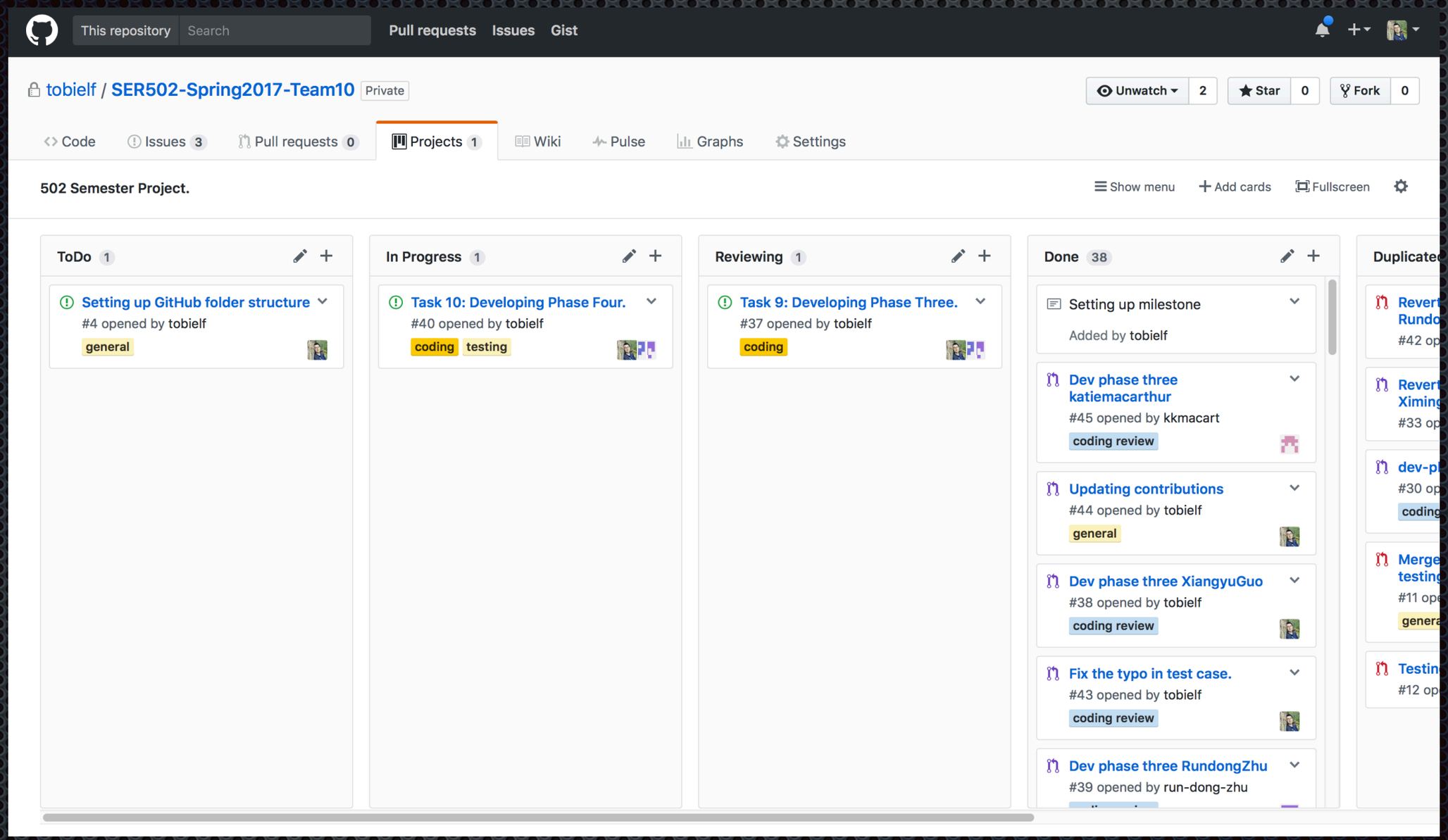
```
Details of the phases:

Saturday - Monday : developing the code.
Monday night      : send out the Pull/Request.
Tuesday noon      : review/change & merge (meeting time)
Tuesday - Thursday: developing the code.
Thursday noon     : resolving problems (meeting time)
Thursday night    : send out the Pull/Request.
Friday            : review/change & merge
```

- Dev phases:
- Dev1: April 15(Sat)-17(Mon), layout the interfaces between components.
- Dev2: April 18(Tue)-21(Fri), implement basic data structure and simple statement.
- Dev3: April 22(Sat)-24(Mon), implement lexical analysis and complex statement.
- Dev4: April 25(Tue)-28(Fri), integrated all parts, testing and fixing bug.

Highlights - Test Driven Dev

- Hand write test case, two cases per person before we start writing the code. (Issue 17, Task 6 Testing Data).
- Unit test on individual component before integrated together.
- Incremental development, developing the project piece by piece and gradually gain the confidence of the code quality.
- Debugging message. In debug mode, useful debugging message will generate on “stderr” to help developer debugging the program, instead of setting up breakpoint and stepping.

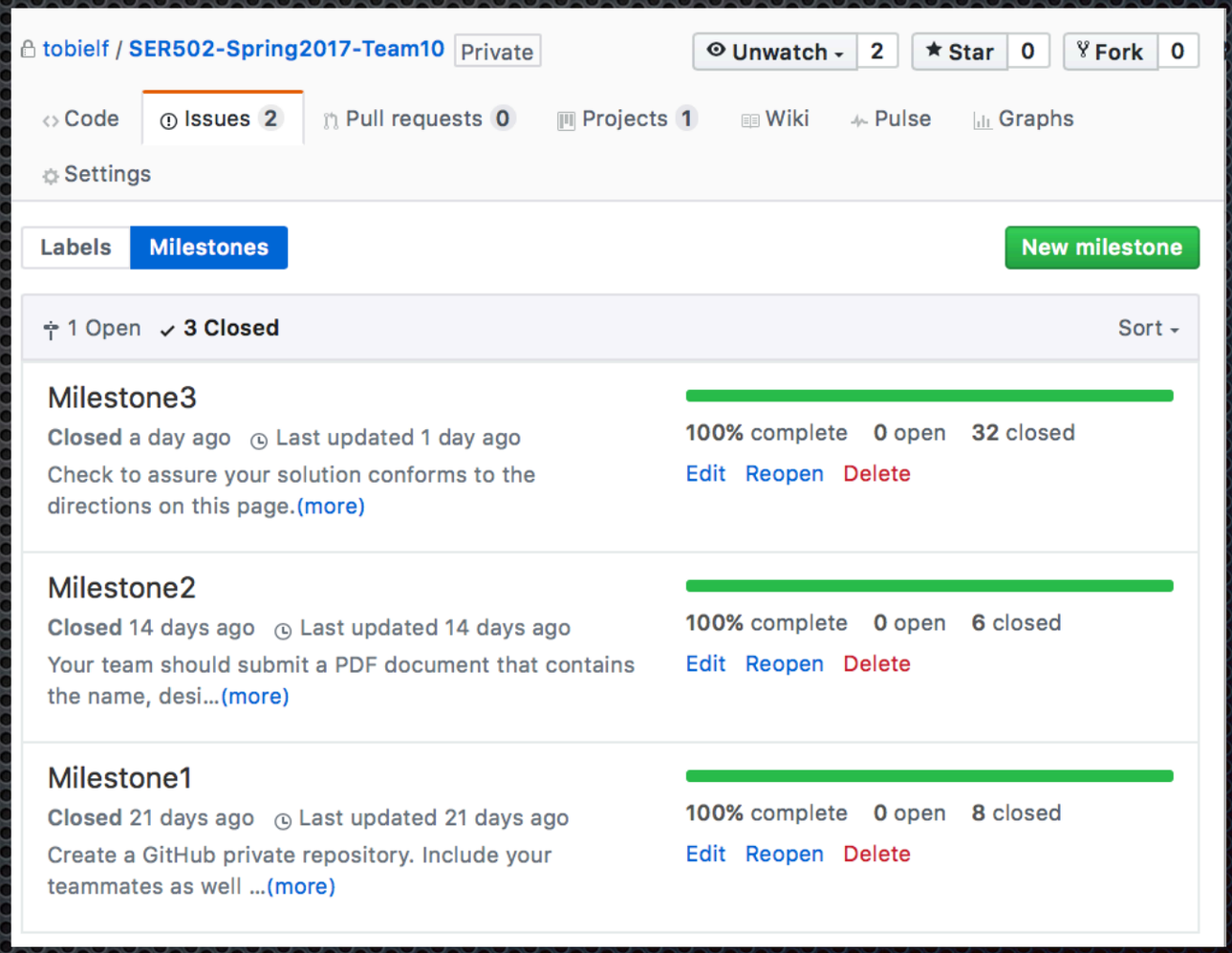
Highlights - Project Manage



- Using GitHub Project Page to organize all tasks in one place.

Highlights - Project Manage

- Setting up every Milestone on GitHub, base on the Project Requirement Document.
- Click the link for more detail inside each Milestone.



Highlights - Issue tracking

- Task 10: Developing Phase Four. coding testing
  #40 by tobielf was closed a day ago Milestone3
- Task 9: Developing Phase Three. coding
  #37 by tobielf was closed 2 days ago Milestone3
- Task 8: Developing Phase Two. coding
  #28 by tobielf was closed 5 days ago Milestone3
- Task 7: Developing Phase One. coding
  #18 by tobielf was closed 8 days ago Milestone3
- Task 6: Testing Data testing
  #17 by tobielf was closed 10 days ago Milestone3
- Task 5: Coding Style. coding
  #14 by tobielf was closed 14 days ago Milestone3
- Task 2: Studying of the prerequisites. general
  #9 by tobielf was closed 19 days ago Milestone2
- Task 1: Get familiar with GitHub general
  #6 by tobielf was closed 22 days ago Milestone1
- Task 4: Language Design and Grammar Rule designing
  #5 by tobielf was closed 14 days ago Milestone2

- Need to add more keywords into symbol table. bug testing
  #61 by tobielf was closed 3 hours ago Final Delivery
- For loop only go upward, can not go downward. bug testing
  #60 by tobielf was closed 2 hours ago Final Delivery
- Endless for loop for reasonable loop incremental expressio
  #59 by tobielf was closed 3 hours ago Final Delivery
- Byte code can not handle boolean value. bug testing
  #58 by tobielf was closed 6 hours ago Final Delivery
- The runtime can not evaluate negative values properly. bug
  #57 by tobielf was closed 21 hours ago Final Delivery
- The compiler can not handle long expression. bug testing
  #56 by tobielf was closed 21 hours ago Final Delivery
- Lexical analysis can't handle negative number properly. bug
  #55 by tobielf was closed 22 hours ago Final Delivery
- Bytecode generator generated duplicated labels and same
  #52 by tobielf was closed 2 days ago Final Delivery
- Parser generated two 'if_stmt' and 'for_stmt' bug testing
  #51 by tobielf was closed 2 days ago Final Delivery

- We tracked all tasks and bugs on GitHub.
- You can go to each issues to view detail discussion history.

Highlights - Code Review

updating-contributio…	Updated 2 days ago by t…	113	0	#54	Merged
dev-phase-four-Xiang…	Updated 2 days ago by t…	81	0	#47	Merged
dev-phase-three-kati…	Updated 3 days ago by t…	90	0	#45	Merged
dev-phase-three-Rund…	Updated 4 days ago by t…	133	0	#43	Merged
dev-phase-two-Ximing	Updated 5 days ago by t…	126	0	#41	Merged
dev-phase-two-Rundon…	Updated 5 days ago by t…	145	0	#35	Merged
dev-phase-three-Xian…	Updated 6 days ago by t…	140	0	#38	Merged
dev-phase-two-katiem…	Updated 7 days ago by t…	193	0	#31	Merged
dev-phase-two-Xiangy…	Updated 7 days ago by t…	186	0	#32	Merged
dev-phase-one-Ximing…	Updated 8 days ago by t…	181	0	#34	Merged
dev-phase-one-Rundon…	Updated 9 days ago by t…	286	0	#26	Merged
dev-phase-one-katiem…	Updated 9 days ago by t…	267	0	#27	Merged
dev-phase-one-Xiangy…	Updated 10 days ago by t…	265	0	#25	Merged
test-case-XimingFeng	Updated 11 days ago by …	264	0	#23	Merged
test-case-rundong	Updated 12 days ago by t…	265	0	#20	Merged
test-case-katiemacar…	Updated 12 days ago by t…	263	0	#24	Merged

Dev phase two ximing coding review
#41 by XimingFeng was merged 5 days ago • Approved Milestone3 — 2

Dev phase three RundongZhu coding review
#39 by run-dong-zhu was merged 4 days ago • Approved Milestone3 — 3

Dev phase three XiangyuGuo coding review
#38 by tobielf was merged 3 days ago Milestone3 — 9

Updating contributions general
#36 by tobielf was merged 5 days ago • Approved Milestone3 — 1

Dev phase two rundongzhu coding review
#35 by run-dong-zhu was merged 5 days ago • Approved Milestone3 — 3

dev-phase-one-XimingFeng coding review
#34 by XimingFeng was merged 8 days ago • Approved Milestone3 — 3

Revert "dev-phase-one-XimingFeng"
#33 by tobielf was merged 8 days ago

Dev phase two XiangyuGuo coding review
#32 by tobielf was merged 7 days ago • Approved Milestone3 — 8

Dev phase two katiemacarthur coding review
#31 by kkmacart was merged 7 days ago • Approved Milestone3 — 5

dev-phase-one-XimingFeng coding review
#30 by XimingFeng was merged 8 days ago • Changes requested Milestone3 — 2

Updating contributions general
#29 by tobielf was merged 9 days ago • Approved Milestone3 — 1

dev-phase-one-katiemacarthur coding review
#27 by kkmacart was merged 9 days ago • Approved Milestone3 — 3

dev-phase-one-Rundong-Zhu coding review
#26 by run-dong-zhu was merged 8 days ago • Approved Milestone3 — 4

dev-phase-one-XiangyuGuo coding review
#25 by tobielf was merged 9 days ago • Approved Milestone3 — 3

Test case katiemacarthur testing
#24 by kkmacart was merged 11 days ago • Approved Milestone3 — 5

- Everyone develop his/her code on independent branches.
- The Pull/Request code will be thoroughly reviewed before merge the code into master.
- Click those links for more detail.

Highlights - Documentation

- Document generated by doxygen directly from the comment section in the code.

Highlights - Memleak free

- As we all know that, the developer needs to handle the Garbage Collection(GC) very carefully in C.
- If you did not free all the dynamic resources you allocated, it’s a memory leakage(Memleak).
- We proudly to announce that our compiler and runtime is 100% Memleak free.

Highlights - Memleak free

```
==6909== Memcheck, a memory error detector
==6909== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==6909== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==6909== Command: ./compiler demo.ten demo.asm
==6909==
--6909-- WARNING: unhandled syscall: 358
--6909-- You may be able to write your own handler.
--6909-- Read the file README_MISSING_SYSCALL_OR_IOCTL.
--6909-- Nevertheless we consider this a bug.  Please report
--6909-- it at http://valgrind.org/support/bug_reports.html.
==6909==
==6909== HEAP SUMMARY:
==6909==     in use at exit: 0 bytes in 0 blocks
==6909==   total heap usage: 432 allocs, 432 frees, 5,419 bytes allocated
==6909==
==6909== All heap blocks were freed -- no leaks are possible
==6909==
==6909== For counts of detected and suppressed errors, rerun with: -v
==6909== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
root@raspberrypi:~/502project/src/compiler#
```

- Memory leak detecting result produced by Valgrind.

Highlights - Memleak free

```
root@raspberrypi:~/502project/src/runtime# valgrind ./runtime demo.asm
==6929== Memcheck, a memory error detector
==6929== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==6929== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==6929== Command: ./runtime demo.asm
==6929==
-1
-100
1
64
==6929==
==6929== HEAP SUMMARY:
==6929==     in use at exit: 0 bytes in 0 blocks
==6929==   total heap usage: 92 allocs, 92 frees, 2,495 bytes allocated
==6929==
==6929== All heap blocks were freed -- no leaks are possible
==6929==
==6929== For counts of detected and suppressed errors, rerun with: -v
==6929== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
root@raspberrypi:~/502project/src/runtime#
```

- Memory leak detecting result produced by Valgrind.

Retrospects - Katie

- At first, I was not familiar with how a symbol table and lexical analyzer worked. But thankfully through conversations with my teammates, I figured it out.
- Also I had trouble with my lexical analyzer code at first, not considering all of the possibilities involved when it took in each char. It worked like a FSM, having to detect what kind of element it was taking in before it would assign a token to it.

Retrospects - Ximing

- My challenge would be the that i was not familiar with how to implement object oriented programming in C which is not designed to be object oriented. I spent a lot of time to understand how it is implemented with the help of pointer to function.
- And also, I haven't use test driven development before, it hard to start with but when i got used to it, I was also amazed by how we build the program so fast, how we cooperated so well. I appreciate each team member's hard working and the help from them. Thank you.

Retrospects - Rundong

- This is the first time I development software in C with Object Oriented Programming thinking.
- Fully understand Test Driven Development with unit-test case implementation.
- Fully understand semantic analysis to figure out when and where to add Bytecode based on Parsing tree. Debug segmentation fault for tree structure.
- Deep insight in how compiler and runtime working through review others code.
- Wonderful team work experience with each team member.

Retrospects - Xiangyu

- For me, programming in C, and developing a simple compiler project is not a big challenge. As well as the Project Management.
- However, I still made some mistakes in the designing of the grammar rule, like it can't handle negative number properly, can't handle long expression and can't make a for loop goes downward. These mistakes almost ruined this project, luckily after three weeks cooperating my team member can resolve these issues in a very short time.
- What I learned from this project is how to dissipate the knowledge across the group, how to assign the proper task to team members based on their skill level. And most importantly how to encourage your teammates so that each team member can made the improvement through the project.
- I am so happy and proud that my team member developed a high quality software and made a great promotion on their skill set.

Future works

- Byte code optimization.
- Improve the symbol table lookup speed.