```
//Sam Dressler
//Header for final code generation using 3 address code and symbol table
#include <stdio.h>
#include <stdlib.h>
#include <vector>
#include <iostream>
#include <stdlib.h>
#include <unistd.h>
#include <bits/stdc++.h>
#include <tuple>
#include <fstream>
#include <regex>
#include <queue>


using namespace std;

//DEFINES
#define BUFF_SIZE 256

//FUNCTIONS
void print_label_to_data_section(FILE *, string line, int mode);
void add_variables_to_data_section(FILE * fcg_out);
void print_message_to_text_section(FILE *fcg_out, string line, int mode);
void initialize_registers();
string get_open_reg(int mode);
void print_registers();
void set_reg_data(string reg, string cmd, int reg_type);
void create_mips_for_assignment(FILE * fcg_out, string line);
void create_mips_for_compound_assign(FILE * fcg_out, string cline, sym_table_entry_t temp_s
ym);
void create_mips_for_single_assign(FILE * fcg_out, string line, sym_table_entry_t temp_sym);
void create_jump_statement(FILE * fcg_out, string line, int mode);
void create_jump_label(FILE * fcg_out, string line);
string get_type_of(string var);
void reset_registers();
//GLOBALS
int curr_reg_num = 0;
int print_label_count = 0;
int print_label_current = 0;
int saved_reg_count = 0;
int saved_reg_current = 0;
int temp_reg_count = 0;
int temp_reg_current = 0;
int labels_count = 0;
int labels_count_at = 0;
int end_if_label_index = 0;
```

```
int else_label_index =0;
string last_v_reg_used;int last_v_reg_used_num;
string last_a_reg_used;int last_a_reg_used_num;
string last_t_reg_used;int last_t_reg_used_num;
string last_s_reg_used;int last_s_reg_used_num;
#define MAX_LABELS 50
string labels[MAX_LABELS];

regex regex_label("(L)([0-9]+)");
//              <reg_name>, <filled_status>,      <data>
typedef tuple<      string,         bool,         string> reg_status_t;
reg_status_t v_registers[2]; //hold instructions
reg_status_t a_registers[2]; //hold addresses
reg_status_t t_registers[10]; //hold temporary information
reg_status_t s_registers[8]; //registers used to save data;

//DRIVER FOR FCG
void fcg_driver(){
   //LOCAL VARIABLES
   FILE * sym_table_in = fopen("output_sym_table.txt","r");
   FILE * fcg_out = fopen("fcg.asm", "w");
   ifstream tac_in ("output_icg.txt");
   string line;
   char delim = '\n';

   //OPEN FILES
   if(!tac_in || !sym_table_in){
      cout << "ERROR : File(s) could not be opened" << endl;
      exit(-1);
   }
   //INITIALIZE REGISTERS
   initialize_registers();

   //begin the data section for print labels
   fprintf(fcg_out, "#Beginning of data section\n.data\n");
      while(std::getline(tac_in, line)){
      if(line.empty())continue;
      if(line.find("writeln") != std::string::npos){
         print_label_to_data_section(fcg_out, line,1);
      }
      else if(line.find("write") != std::string::npos){
         print_label_to_data_section(fcg_out, line,2);
      }
      else if(regex_match(line, regex_label)){
         create_jump_label(fcg_out, line);
      }
```

```
    else if(line.compare("else")==0){
        else_label_index = labels_count;
    }
    else if(line.compare("end_if")==0){
        end_if_label_index = labels_count;
    }
}
add_variables_to_data_section(fcg_out);
//REFRESH FILE POINTER
tac_in.close();
tac_in.open("output_icg.txt");

//begin the text section which will contain the code
fprintf(fcg_out, "#Beginning of Code section\n.text\nstart:\n");
while(std::getline(tac_in, line)){
    // cout << "LINE : " << line <<"!" << endl;
    if(line.empty())continue;
    if(line.find("writeln")!= std::string::npos){
        print_message_to_text_section(fcg_out, line, 1);
    }
    else if(line.find("write")!= std::string::npos){
        print_message_to_text_section(fcg_out, line, 2);
    }
    else if(line.find(":=") != std::string::npos){
        create_mips_for_assignment(fcg_out, line);
    }
    else if(line.compare("jump else")==0){
        fprintf(fcg_out, "j %s\n\n",labels[else_label_index].c_str());
    }
    else if(line.compare("jump end_if") == 0){
        fprintf(fcg_out, "j %s\n\n",labels[end_if_label_index].c_str());
    }
    else if(line.find("if") != string::npos){
        create_jump_statement(fcg_out, line, 1);
    }
    else if(regex_match(line, regex_label)){
        fprintf(fcg_out, "%s:\n",labels[labels_count_at].c_str());
        labels_count_at++;
    }



}

fprintf(fcg_out, "li $v0, 10\n");//exit command
fprintf(fcg_out, "syscall\n\n");
// print_registers();
```

```
    fclose(sym_table_in);
    tac_in.close();
    fclose(fcg_out);
}
void create_jump_statement(FILE *fcg_out, string line, int mode){
    size_t pos;
    string var1;
    string var2;
    string var1_reg;
    string var2_reg;
    string op;
    switch(mode){
        case 1: // for if statements
            pos = line.find_first_of(" ");
            line = line.erase(0, pos+1); // erase the if keyword
            pos = line.find_first_of(" ");
            var1 = line.substr(0, pos);
            line = line.erase(0, pos+1);
            pos = line.find_first_of(" ");
            op = line.substr(0, pos+1);
            line = line.erase(0, pos-1);
            pos = line.find_first_of(" ");
            var2 = line.substr(pos+1, line.length());
            var1_reg = get_open_reg(3);
            var2_reg = get_open_reg(3);
            op = trim(op);
            if(op.compare("=")==0){
                fprintf(fcg_out, "lw $%s, %s\n",var1_reg.c_str(), var1.c_str());
                fprintf(fcg_out, "lw $%s, %s\n",var2_reg.c_str(), var2.c_str());
                fprintf(fcg_out, "beq $%s, $%s, %s\n\n",var1_reg.c_str(), var2_reg.c_str(), labels[label
s_count_at].c_str());
            }
            else if(op.compare("<")==0){
                fprintf(fcg_out, "lw $%s, %s\n",var1_reg.c_str(), var1.c_str());
                fprintf(fcg_out, "lw $%s, %s\n",var2_reg.c_str(), var2.c_str());
                fprintf(fcg_out, "blt $%s, $%s, %s\n\n",var1_reg.c_str(), var2_reg.c_str(), labels[labels
_count_at].c_str());
            }
            else if(op.compare(">")==0){
                fprintf(fcg_out, "lw $%s, %s\n",var1_reg.c_str(), var1.c_str());
                fprintf(fcg_out, "lw $%s, %s\n",var2_reg.c_str(), var2.c_str());
                fprintf(fcg_out, "bgt $%s, $%s, %s\n\n",var1_reg.c_str(), var2_reg.c_str(), labels[label
s_count_at].c_str());
            }
            else if(op.compare("<=")==0){
```

```
            fprintf(fcg_out, "lw $%s, %s\n",var1_reg.c_str(), var1.c_str());
            fprintf(fcg_out, "lw $%s, %s\n",var2_reg.c_str(), var2.c_str());
            fprintf(fcg_out, "ble $%s, $%s, %s\n\n",var1_reg.c_str(), var2_reg.c_str(),labels[labels
_count_at].c_str());
        }
        else if(op.compare(">=")==0){
            fprintf(fcg_out, "lw $%s, %s\n",var1_reg.c_str(), var1.c_str());
            fprintf(fcg_out, "lw $%s, %s\n",var2_reg.c_str(), var2.c_str());
            fprintf(fcg_out, "bge $%s, $%s, %s\n\n",var1_reg.c_str(), var2_reg.c_str(), labels[label
s_count_at].c_str());
        }
      break;
      case 2:
      break;
      default:
      break;
   }
   reset_registers();
   return;
}
void create_jump_label(FILE * fcg_out, string line){
   labels[labels_count] = line;
   labels_count++;
   ; // push the label into the queue to be popped
}

void create_mips_for_assignment(FILE * fcg_out, string line){
   sym_table_entry_t temp_sym;
   string curr_var;
   string curr_var_val;
   string entry_id;
   bool compound_val = false;
   bool single_val = false;
   size_t pos = line.find_first_of(":=");
   curr_var = line.substr(0,pos-1);
   // cout << "curr_var : " << curr_var << endl;
   curr_var_val = line.erase(0, pos+3);
   if((pos = curr_var_val.find_first_of("+*-/")) != string::npos){
      curr_var_val = line;
      // cout << "COMPOUND VAL : " << curr_var_val << endl;
      compound_val = true;
   }
   else{
      curr_var_val = line;
      // cout << "CURR VAL : " << curr_var_val << endl;
      single_val = true;
```

```
    }
    for(vector<sym_table_entry_t>::iterator i = sym_table.begin(); i < sym_table.end(); i++){
        temp_sym = *i;
        entry_id = get<0>(temp_sym);
        if(entry_id.compare(curr_var)==0){
            if(single_val){
                create_mips_for_single_assign(fcg_out, line, temp_sym);
            }else if(compound_val){
                create_mips_for_compound_assign(fcg_out, line, temp_sym);
            }
        }
        else{
            continue;
        }
    }
}
void create_mips_for_single_assign(FILE * fcg_out, string line, sym_table_entry_t temp_sym){
    size_t pos;
    string entry_id;
    string entry_data;
    string entry_type;
    string temp_reg;
    string open_reg;
    entry_id = get<0>(temp_sym);
    entry_data = get<3>(temp_sym);
    entry_type = get<4>(temp_sym);
    if(entry_type.compare("integer")==0){
        if((pos = entry_id.find("[")) != string::npos){
            //remove brackets from var name to create legal mips syntax
            entry_id.at(pos) = entry_id.at(pos+1);
            entry_id.erase(pos+1, entry_id.length());
        }
        if(line.find("read") != string::npos){
            //found read so store return from readln in temp reg, then store temp reg in source
            temp_reg = get_open_reg(1);
            fprintf(fcg_out, "li $%s, 5\n", temp_reg.c_str()); //5 is command for readln
            fprintf(fcg_out, "syscall \n\n"); // v0 is always the return register for readln int
            fprintf(fcg_out, "sw $v0, %s\n\n", entry_id.c_str());
        }
        else{

            saved_reg_count++;
            open_reg = get_open_reg(4);//find an s reg
            // Store open reg as the destination(store) reg
            fprintf(fcg_out, "lw $%s, %s\n",open_reg.c_str(), entry_id.c_str()); //store RAM _SOURC
E into REGISTER DEST
```

```
            open_reg = get_open_reg(3);
            temp_reg = open_reg;
            entry_data = trim(entry_data);
            if(regex_match(entry_data, regex_number)){
               fprintf(fcg_out, "li $%s, %s\n",open_reg.c_str(), entry_data.c_str()); // Store value to r
egister
            }else{
               fprintf(fcg_out, "lw $%s, %s\n",open_reg.c_str(), entry_data.c_str()); // Store value to
register
            }
            fprintf(fcg_out, "sw $%s, %s\n\n",temp_reg.c_str(), entry_id.c_str()); //register source to
ram destination
         }
      }
      else if(entry_type.compare("char")==0){
         temp_reg_count++;
         open_reg = get_open_reg(4);
         fprintf(fcg_out, "lb $%s, %s\n",open_reg.c_str(), entry_id.c_str());
         open_reg = get_open_reg(3);
         temp_reg = open_reg;
         fprintf(fcg_out, "li $%s, \'%s\'\n",open_reg.c_str(), entry_data.c_str()); // Store value to regi
ster
         fprintf(fcg_out, "sb $%s, %s\n\n",temp_reg.c_str(), entry_id.c_str()); //register source to ra
m destination
      }
   }
   void create_mips_for_compound_assign(FILE * fcg_out, string cline, sym_table_entry_t temp_s
   ym){
      string var1;
      string var2;
      string temp_reg1;
      string temp_reg2;
      string op_dest_reg;
      string op;
      bool is_constant = false;
      string dest_var_name = get<0>(temp_sym);
      size_t pos = cline.find_first_of("+-*/");
      op = cline.at(pos);
      var1 = cline.substr(0,pos-1);
      var2 = cline.substr(pos+2, cline.length());
      if((pos = var1.find("[")) != string::npos){
         //remove brackets from var name to create legal mips syntax
         var1.at(pos) = var1.at(pos+1);
         var1.erase(pos+1, var1.length());
      }
      if((pos = var2.find("[")) != string::npos){
```

```
        //remove brackets from var name to create legal mips syntax
        var2.at(pos) = var2.at(pos+1);
        var2.erase(pos+1, var2.length()-1);
    }
    var2 = trim(var2);
    if(regex_match(var2, regex_number)){
        is_constant = true;
    }

    // cout << "V1 :" << var1 << " V2:" << var2 << endl;
    // cout << "OP :" << op << endl;
        if(op == "+"){
            temp_reg1 = get_open_reg(3);
            fprintf(fcg_out, "lw $%s, %s\n",temp_reg1.c_str(), var1.c_str());
            temp_reg2 = get_open_reg(3);
            if(is_constant){
                fprintf(fcg_out, "li $%s, %s\n",temp_reg2.c_str(), var2.c_str());
            }
            else{
                fprintf(fcg_out, "lw $%s, %s\n",temp_reg2.c_str(), var2.c_str());
            }
            op_dest_reg = get_open_reg(3);
            fprintf(fcg_out, "add $%s, $%s, $%s\n",op_dest_reg.c_str(), temp_reg1.c_str(), temp_reg
2.c_str());
            fprintf(fcg_out, "sw $%s, %s\n\n", op_dest_reg.c_str(), dest_var_name.c_str());
        }
        else if(op == "-"){
            temp_reg1 = get_open_reg(3);
            fprintf(fcg_out, "lw $%s, %s\n",temp_reg1.c_str(), var1.c_str());
            temp_reg2 = get_open_reg(3);
            if(is_constant){
                fprintf(fcg_out, "li $%s, %s\n",temp_reg2.c_str(), var2.c_str());
            }
            else{
                fprintf(fcg_out, "lw $%s, %s\n",temp_reg2.c_str(), var2.c_str());
            }
            op_dest_reg = get_open_reg(3);
            fprintf(fcg_out, "sub $%s, $%s, $%s\n",op_dest_reg.c_str(), temp_reg1.c_str(), temp_reg
2.c_str());
            fprintf(fcg_out, "sw $%s, %s\n\n", op_dest_reg.c_str(), dest_var_name.c_str());
        }
        else if(op == "*"){
            temp_reg1 = get_open_reg(3);
            fprintf(fcg_out, "lw $%s, %s\n",temp_reg1.c_str(), var1.c_str());
            temp_reg2 = get_open_reg(3);
            if(is_constant){
```

```
            fprintf(fcg_out, "li $%s, %s\n",temp_reg2.c_str(), var2.c_str());
        }
        else{
            fprintf(fcg_out, "lw $%s, %s\n",temp_reg2.c_str(), var2.c_str());
        }
        op_dest_reg = get_open_reg(3);
        fprintf(fcg_out, "mult $%s, $%s\n", temp_reg1.c_str(), temp_reg2.c_str());
        fprintf(fcg_out, "mflo $%s\n", op_dest_reg.c_str());
        fprintf(fcg_out, "sw $%s, %s\n\n", op_dest_reg.c_str(), dest_var_name.c_str());
    }
    else if(op == "/"){
        temp_reg1 = get_open_reg(3);
        fprintf(fcg_out, "lw $%s, %s\n", temp_reg1.c_str(), var1.c_str());
        temp_reg2 = get_open_reg(3);
        if(is_constant){
            fprintf(fcg_out, "li $%s, %s\n",temp_reg2.c_str(), var2.c_str());
        }
        else{
            fprintf(fcg_out, "lw $%s, %s\n",temp_reg2.c_str(), var2.c_str());
        }
        op_dest_reg = get_open_reg(3);
        fprintf(fcg_out, "div $%s, $%s\n", temp_reg1.c_str(), temp_reg2.c_str());
        fprintf(fcg_out, "mfhi $%s", op_dest_reg.c_str()); // Only moving integer quotient to resu
lt register;
        fprintf(fcg_out, "sw $%s, %s\n\n", op_dest_reg.c_str(), dest_var_name.c_str());
    }
    else{
        cout << "ERROR : Invalid Operator\n";
        exit(-1);
    }
}
void add_variables_to_data_section(FILE * fcg_out){
    sym_table_entry_t temp_entry;
    size_t pos;
    size_t pos_end;
    size_t count = 0;
    string entry;
    string entry_type;
    for(vector<sym_table_entry_t>::iterator i = sym_table.begin(); i < sym_table.end(); i++){
        temp_entry = *i;
        entry = get<0>(temp_entry);
        entry_type = get<4>(temp_entry);
        if(entry_type.compare("integer")==0){
            // cout <<"Integer : " << entry << endl;
            if((pos = entry.find("[")) != string::npos){
                //remove brackets from var name to create legal mips syntax
```

```
            entry.at(pos) = entry.at(pos+1);
            entry.erase(pos+1, entry.length());
        }
        fprintf(fcg_out, "%s : .word 0\n", entry.c_str());
    }
    else if(entry_type.compare("char") == 0){
        // cout <<"Char : "<< entry << endl;
        fprintf(fcg_out, "%s : .byte 0\n", entry.c_str());
    }
  }
  return;
}
                                //mode
void set_reg_data(string reg, string cmd, int reg_type){
  reg_status_t temp_reg;
  string name,data;
  bool status;
  switch(reg_type){
    case 1 ://v type
    for(int i = 0; i < 2; i ++){
        temp_reg = v_registers[i];
        if(get<0>(temp_reg).compare(reg)==0){
            name = get<0>(temp_reg);
            status = get<1> (temp_reg);
            data = cmd;
            temp_reg = make_tuple(name, status, data);
            v_registers[i] = temp_reg;
        }
    }
    break;
    case 2 ://a type
    for(int i = 0; i < 2; i ++){
        temp_reg = a_registers[i];
        if(get<0>(temp_reg).compare(reg)==0){
            name = get<0>(temp_reg);
            status = get<1> (temp_reg);
            data = cmd;
            temp_reg = make_tuple(name, status, data);
            a_registers[i] = temp_reg;
        }
    }
    break;
    default:

    break;
  }
```

```
}
//returns an open register of the specified type. If none are open it will use the register that has n
ot been used most recently
string get_open_reg(int mode){
    reg_status_t temp_reg1;
    string name;
    bool status;
    string data;
    switch(mode){
        case 1:
            for(int i = 0; i < 2; i ++){
                temp_reg1 = v_registers[i];
                if(get<1>(temp_reg1) == false){
                    last_v_reg_used = "v"+to_string(i);
                    last_v_reg_used_num = i;
                    name = get<0>(temp_reg1);
                    status = true;
                    data = get<2>(temp_reg1);
                    temp_reg1 = make_tuple(name, status, data);

                    v_registers[i] = temp_reg1;
                    return get<0>(temp_reg1);
                }
                else{
                    continue;
                }
            }
            //no registers available so use the register that has been used te longest ago.
            for(int i = 0; i < 2; i++){
                if(i != last_v_reg_used_num){
                    //Set the previous used register to false so it is freed up
                    temp_reg1 = v_registers[last_v_reg_used_num];
                    name = last_v_reg_used;status = false; data = "empty";
                    v_registers[last_v_reg_used_num] = make_tuple(name, status, data);
                    //do normal task
                    temp_reg1 = v_registers[i];
                    last_v_reg_used = "v"+to_string(i);
                    last_v_reg_used_num = i;
                    name = get<0>(temp_reg1);
                    status = true;
                    data = get<2>(temp_reg1);
                    temp_reg1 = make_tuple(name, status, data);
                    v_registers[i] = temp_reg1;
                    return get<0>(temp_reg1);
                }
            }
```

```
        break;
    case 2:
        for(int i = 0; i < 2; i ++){
            temp_reg1 = a_registers[i];
            if(get<1>(temp_reg1) == false){
                last_a_reg_used = "a"+to_string(i);
                last_a_reg_used_num = i;
                name = get<0>(temp_reg1);
                status = true;
                data = get<2>(temp_reg1);
                temp_reg1 = make_tuple(name, status, data);

                a_registers[i] = temp_reg1;
                return get<0>(temp_reg1);
            }
            else{
                continue;
            }
        }
        for(int i = 0; i < 2; i++){
            if(i != last_a_reg_used_num){
                //Set the previous used register to false so it is freed up
                temp_reg1 = a_registers[last_a_reg_used_num];
                name = last_a_reg_used;status = false; data = "empty";
                a_registers[last_a_reg_used_num] = make_tuple(name, status, data);
                //do normal task
                temp_reg1 = a_registers[i];
                last_a_reg_used = "a"+to_string(i);
                last_a_reg_used_num = i;
                name = get<0>(temp_reg1);
                status = true;
                data = get<2>(temp_reg1);
                temp_reg1 = make_tuple(name, status, data);
                a_registers[i] = temp_reg1;
                return get<0>(temp_reg1);
            }
        }
    break;
case 3:
        for(int i = 0; i < 10; i ++){
            temp_reg1 = t_registers[i];
            if(get<1>(temp_reg1) == false){
                last_t_reg_used = "t"+to_string(i);
                last_t_reg_used_num = i;
                name = get<0>(temp_reg1);
                status = true;
```

```
            data = get<2>(temp_reg1);
            temp_reg1 = make_tuple(name, status, data);

            t_registers[i] = temp_reg1;
            return get<0>(temp_reg1);
        }
        else{
            continue;
        }
    }
    for(int i = 0; i < 10; i++){
        if(i != last_t_reg_used_num){
            //Set the previous used register to false so it is freed up
            temp_reg1 = t_registers[last_t_reg_used_num];
            name = last_t_reg_used;status = false; data = "empty";
            t_registers[last_t_reg_used_num] = make_tuple(name, status, data);
            //do normal task
            temp_reg1 = t_registers[i];
            last_t_reg_used = "t"+to_string(i);
            last_t_reg_used_num = i;
            name = get<0>(temp_reg1);
            status = true;
            data = get<2>(temp_reg1);
            temp_reg1 = make_tuple(name, status, data);
            t_registers[i] = temp_reg1;
            return get<0>(temp_reg1);
        }
    }
break;
case 4:
    for(int i = 0; i < 8; i ++){
        temp_reg1 = s_registers[i];
        if(get<1>(temp_reg1) == false){
            last_s_reg_used = "s"+to_string(i);
            last_s_reg_used_num = i;
            name = get<0>(temp_reg1);
            status = true;
            data = get<2>(temp_reg1);
            temp_reg1 = make_tuple(name, status, data);

            s_registers[i] = temp_reg1;
            return get<0>(temp_reg1);
        }
        else{
            continue;
        }
```

```
            }
            for(int i = 0; i < 8; i++){
                if(i != last_s_reg_used_num){
                    //Set the previous used register to false so it is freed up
                    temp_reg1 = s_registers[last_s_reg_used_num];
                    name = last_s_reg_used;status = false; data = "empty";
                    s_registers[last_s_reg_used_num] = make_tuple(name, status, data);
                    //do normal task
                    temp_reg1 = s_registers[i];
                    last_s_reg_used = "s"+to_string(i);
                    last_s_reg_used_num = i;
                    name = get<0>(temp_reg1);
                    status = true;
                    data = get<2>(temp_reg1);
                    temp_reg1 = make_tuple(name, status, data);
                    s_registers[i] = temp_reg1;
                    return get<0>(temp_reg1);
                }
            }
        break;
        default:
        break;
    }
    //TODO HANDLE THE CASE WHERE ALL ARE FULL

}

//print labels for writeln to data section that will be printed later
void print_label_to_data_section(FILE * fcg_out, string line, int mode){
    size_t pos;
    size_t pos1;
    size_t pos2;
    pos1 = line.find_first_of("\"");
    pos2 = line.find_first_of("\"");
    if(pos1 == string::npos && pos2 == string::npos){
        if(mode==1){
            mode = 3; // handle printing variables plus newline
        }else if(mode == 2) {
            mode = 4; //hanlde just printing variables
        }
    }
    else{
        print_label_count ++;
        pos = line.find_first_of(' ');
        line.erase(0, pos+1);
        line.at(0) = '\"';
```

```
    }
    //do specific stuff for write vs writeln
    switch(mode){
        case 1:
            line.at(line.length()-1) = '\\';
            line.append("n\"");
            fprintf(fcg_out,"PL%d : .asciiz %s\n",print_label_count, line.c_str());
        break;
        case 2:
            line.at(line.length()-1) = '\"';
            fprintf(fcg_out,"PL%d : .asciiz %s\n",print_label_count, line.c_str());
        break;
        case3:
        case4:
            //do nothing for now since we don't need a label to print a variable
        break;
        default:
        break;
    }
}
string get_type_of(string var){
    sym_table_entry_t temp_var;
    var = trim(var);
    for(vector<sym_table_entry_t>::iterator i = sym_table.begin(); i <sym_table.end(); i++){
        temp_var = *i;
        if(get<0>(temp_var).compare(var)==0){
            return get<4>(temp_var);
        }
        else{continue;}
    }
    cout << "ERROR (final_code_generator): undefined variable" << endl;
    exit(-1);
}
void print_message_to_text_section(FILE *fcg_out,string line, int mode){
    size_t pos;
    size_t pos1;
    size_t pos2;
    string type;
    string open_reg;
    string label;
    pos1 = line.find_first_of("\"");
    pos2 = line.find_first_of("\"");
    if(pos1 == string::npos && pos2 == string::npos){
        if(mode==1){
            mode = 3; // handle printing variables plus newline
        }else if(mode == 2) {
```

```
        mode = 4; //hanlde just printing variables
    }
}
else{
    print_label_count ++;
    pos = line.find_first_of(' ');
    line.erase(0, pos+1);
    line.at(0) = '\"';
}

switch(mode){
    case 1:
        open_reg = "v0"; // v0 must contina command
        set_reg_data(open_reg, to_string(4), 1);
        fprintf(fcg_out, "li $v0, 4\n");
        print_label_current++;
        label = "PL" + to_string(print_label_current);
        fprintf(fcg_out, "la $a0, %s\n", label.c_str());
        set_reg_data(open_reg, label, 2);
        fprintf(fcg_out, "syscall\n\n");
    break;

    case 2:
        open_reg = "v0"; // v0 must contain command
        set_reg_data(open_reg, to_string(4), 1);
        fprintf(fcg_out, "li $v0, 4\n");
        print_label_current++;
        label = "PL" + to_string(print_label_current);
        fprintf(fcg_out, "la $a0, %s\n", label.c_str());
        set_reg_data(open_reg, label, 2);
        fprintf(fcg_out, "syscall\n\n");
    break;

    case 3:
        line.erase(0, 8); //erase writeln
        type = get_type_of(line);
        open_reg = get_open_reg(3);
        if(type == "integer"){
            fprintf(fcg_out, "li $v0, 1\n"); //command for printing integer
            fprintf(fcg_out, "lw $%s, %s\n",open_reg.c_str(), line.c_str());
            fprintf(fcg_out, "move $a0, $%s\n",open_reg.c_str());
            fprintf(fcg_out, "syscall\n\n");
        }
        else if(type == "char"){
            fprintf(fcg_out, "li $v0, 11\n"); //command for printing character
            fprintf(fcg_out, "lb $%s, %s\n",open_reg.c_str(), line.c_str());
```

```
            fprintf(fcg_out, "move $a0, $%s\n",open_reg.c_str());
            fprintf(fcg_out, "syscall\n\n");

        }

      break;
      case 4:
         line.erase(0, 6); //erase write
      break;
   }
}
//generate the register names for each of the 4 types of registers
void initialize_registers(){
   string reg_name;
   for(int i = 0; i < 2; i ++){
      reg_name = "v" + to_string(i);
      reg_status_t temp_reg = make_tuple(reg_name, false, "empty");
      v_registers[i] = temp_reg;
      // cout << get<0>(v_registers[i])<<endl;
   }
   for(int i = 0; i < 2; i ++){
      reg_name = "a" + to_string(i);
      reg_status_t temp_reg = make_tuple(reg_name, false, "empty");
      a_registers[i] = temp_reg;
   }
   for(int i = 0; i < 10; i ++){
      reg_name = "t" + to_string(i);
      reg_status_t temp_reg = make_tuple(reg_name, false, "empty");
      t_registers[i] = temp_reg;
   }
   for(int i = 0; i < 8; i ++){
      reg_name = "s" + to_string(i);
      reg_status_t temp_reg = make_tuple(reg_name, false, "empty");
      s_registers[i] = temp_reg;
   }
   return;
}
void reset_registers(){
   string reg_name;
   for(int i = 0; i < 2; i ++){
      reg_name = "v" + to_string(i);
      reg_status_t temp_reg = make_tuple(reg_name, false, "empty");
      v_registers[i] = temp_reg;
      // cout << get<0>(v_registers[i])<<endl;
   }
   for(int i = 0; i < 2; i ++){
```

```
        reg_name = "a" + to_string(i);
        reg_status_t temp_reg = make_tuple(reg_name, false, "empty");
        a_registers[i] = temp_reg;
    }
    for(int i = 0; i < 10; i ++){
        reg_name = "t" + to_string(i);
        reg_status_t temp_reg = make_tuple(reg_name, false, "empty");
        t_registers[i] = temp_reg;
    }
    for(int i = 0; i < 8; i ++){
        reg_name = "s" + to_string(i);
        reg_status_t temp_reg = make_tuple(reg_name, false, "empty");
        s_registers[i] = temp_reg;
    }
}

void print_registers(){
    string status;
    cout << "----------------------------------------"<<endl;
    cout << "               REGISTERS " << endl;
    cout << "----------------------------------------"<<endl;
    cout << "       LAST V REG USED : " << last_v_reg_used << endl;
    reg_status_t temp_reg;
        for(int i = 0; i < 2; i ++){
        temp_reg = v_registers[i];
        status = (get<1>(temp_reg) ? "true" : "false");
        printf("%-5s | %-
7s | %10s\n", get<0>(temp_reg).c_str(), status.c_str(), get<2>(temp_reg).c_str());
    }
    cout << "----------------------------------------"<<endl;
    cout << "       LAST A REG USED : " << last_a_reg_used << endl;
    for(int i = 0; i < 2; i ++){
        temp_reg = a_registers[i];
        status = (get<1>(temp_reg) ? "true" : "false");
        printf("%-5s | %-
7s | %10s\n", get<0>(temp_reg).c_str(), status.c_str(), get<2>(temp_reg).c_str());
    }
    cout << "----------------------------------------"<<endl;
    cout << "       LAST T REG USED : " << last_t_reg_used << endl;
    for(int i = 0; i < 10; i ++){
        temp_reg = t_registers[i];
        status = (get<1>(temp_reg) ? "true" : "false");
        printf("%-5s | %-
7s | %10s\n", get<0>(temp_reg).c_str(), status.c_str(), get<2>(temp_reg).c_str());
    }
    cout << "----------------------------------------"<<endl;
```

```
    cout << "       LAST S REG USED : " << last_s_reg_used << endl;
    for(int i = 0; i < 8; i ++){
        temp_reg = s_registers[i];
        status = (get<1>(temp_reg) ? "true" : "false");
        printf("%-5s | %-
7s | %10s\n", get<0>(temp_reg).c_str(), status.c_str(), get<2>(temp_reg).c_str());
    }
    return;
}
```