```
// To parse variables
// map<string, int> location; // can use unordered or ordered
maps
// var_section → id_list SEMICOLON
// id_list → ID COMMA id list | ID
void parse_id_list(){
     // expect ID

     // set location of ID as next available location in map
     location[“The lexeme for the ID”] = next_available
     // in complier.h
     mem[next_available] = 0;
     next_available++;
     // you can use a if here
     // look for COMMA. Why? Yes, peek()
     // if found what should we do? Call parse_id_list again
}
```

Example:

```
// a, b, c, d;
// Assigning location for variable "a" | Token = ID
     int address_a = next_available;
    mem[next_available] = 0;
    next_available++;

// Assigning location for variable "b"
     int address_b = next_available;
    mem[next_available] = 0;
     next_available++;
```

```
// To parse Inputs
// The list of input values is called inputs and appears as the
last section of the input to your compiler. This list must be
read by your compiler and stored in an inputs array, which is
simply a vector of integers

// inputs → num_list
// num_list → NUM
// num_list → NUM num_list

void parse_num_list(){
     // expect NUM
     // declared in compiler.h
     // coversion using stoi is necessary
     // inputs.pushback("The lexeme for the NUM")
     // check if another NUM exist | use peek
     // if yes what to do? Call parse_num_list again
}

Example:
// 1 2 3 4 5 6
// Inputs
     inputs.push_back(1); // Token = NUM
     inputs.push_back(2);
     inputs.push_back(3);
     inputs.push_back(4);
     inputs.push_back(5);
     inputs.push_back(6);
```

```
//To parse IF statements
//if_stmt → IF condition body

void parse_if_stmt(){
     // expect IF
     // create instruction node
     // set its type as CJMP right? yes

     // create another instruction node
    // what is the type? noop


     // what is the next thing?
     // condition → primary relop primary | c <> a
     // parse_condition
     // what should parse condition do?
     // set the other parameters of if-stmt
    // struct CJMP {
    //     ConditionalOperatorType condition_op;
    //     int operand1_index;
    //     int operand2_index;
    //     struct InstructionNode * target;
     //  }
     // new instruction node
     // parse_body
     // new instruction node should be holding what? body
     // where should noop node be appended? After the body

     // where should noop be targeted? In the target of if
     // where should node holding body be appended? After the if

     Return instruction node of if
}

Example:
//   IF c <> a (i4)
//   {
//        output b; (i5)
//   } noop (i6)
struct InstructionNode * i4 = new InstructionNode;
struct InstructionNode * i5 = new InstructionNode;
```

```
struct InstructionNode * i6 = new InstructionNode;

i6->type = NOOP;                                        // NOOP
after IF
i6->next = NULL;
i4->type = CJMP;                                  // if c <> a
i4->cjmp_inst.condition_op = CONDITION_NOTEQUAL;
i4->cjmp_inst.operand1_index = address_c; location[“c”]
i4->cjmp_inst.operand2_index = address_a; location[“a”]
i4->cjmp_inst.target = i6;                        // if not (c <> a)
skip forward to NOOP

i4->next = i5;

i5->type = OUT;                                        // output b
i5->output_inst.var_index = address_b;
i5->next = i6;
```