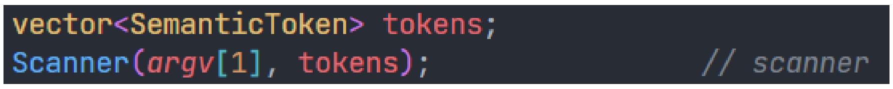
CSC4180-a4-report

Scanner

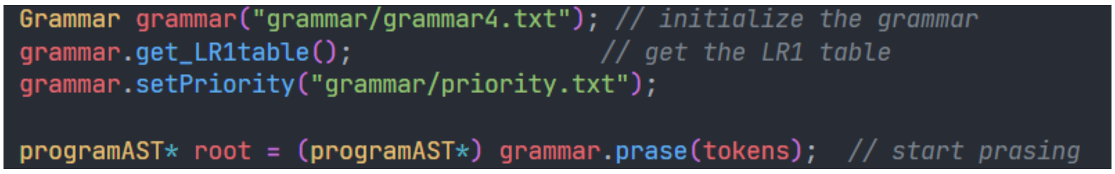
I use the scanner written in Assignment2 and motify a bit. It accepts a file and translate the file into a vector of semantic record, like {INTNUM, 10000}. Then the parser uses this vector to do the parsing. I use a finite state machine to achieve the scanner.



```
vector<SemanticToken> tokens;
Scanner(argv[1], tokens);                    // scanner
```

Parser

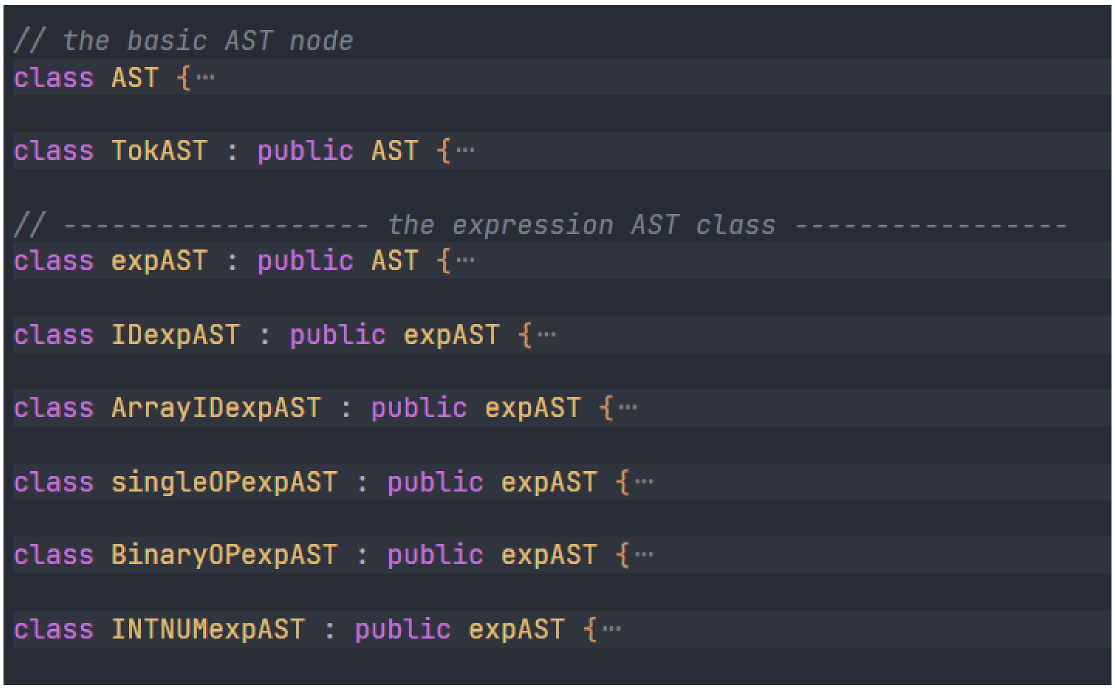
I use the parser written in Assignment3 and motify a bit. It accepts a vector of tokens and build a AST tree for the code generator. The AST tree is defined in AST.h.



```
Grammar grammar("grammar/grammar4.txt"); // initialize the grammar
grammar.get_LR1table();                  // get the LR1 table
grammar.setPriority("grammar/priority.txt");

programAST* root = (programAST*) grammar.prase(tokens);  // start prasing
```

Parsing process



```
// the basic AST node
class AST {…

class TokAST : public AST {…

// ------------------- the expression AST class -----------------
class expAST : public AST {…

class IDexpAST : public expAST {…

class ArrayIDexpAST : public expAST {…

class singleOPexpAST : public expAST {…

class BinaryOPexpAST : public expAST {…

class INTNUMexpAST : public expAST {…
```

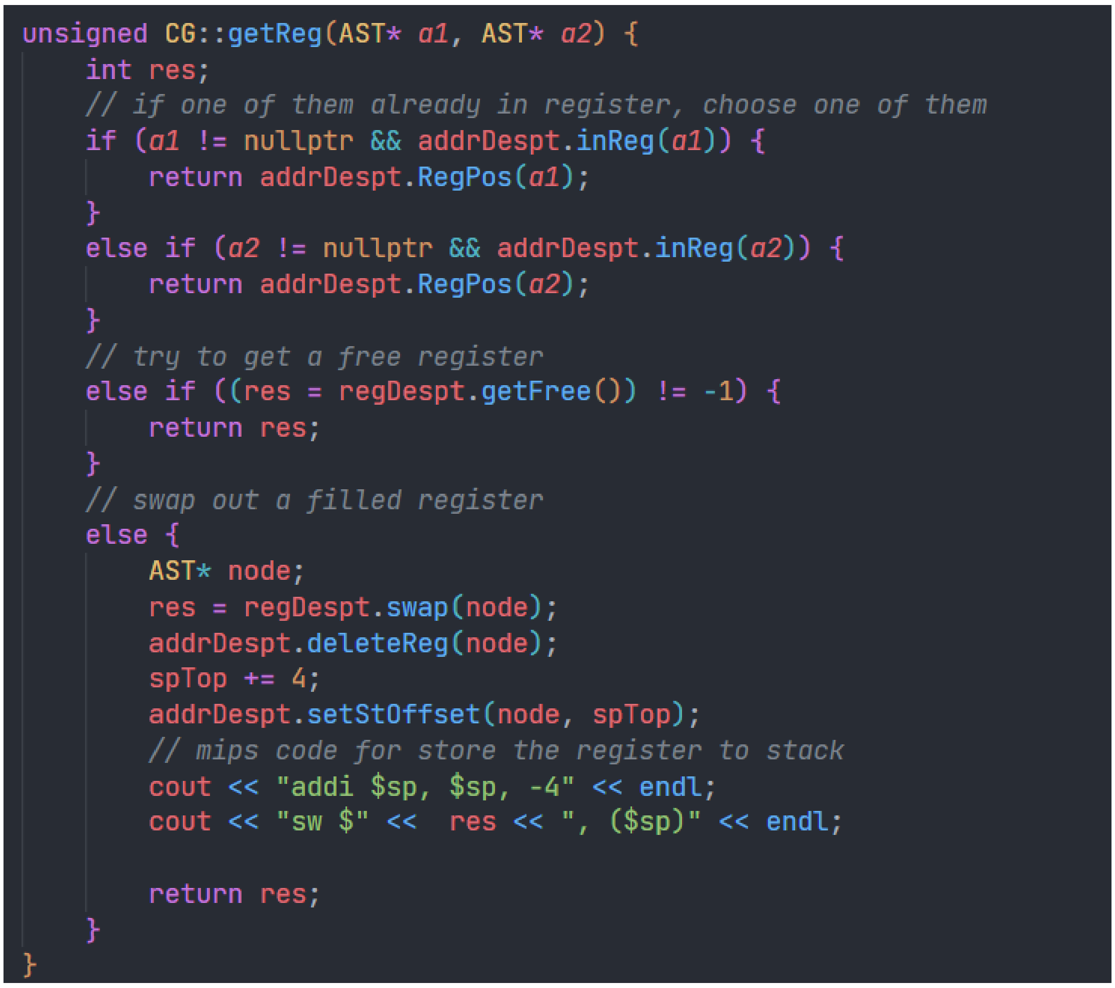
Part of expression AST node class

I also do some optimization in the parser. For example, when reducing 2 expression node to 1, if both of them are INT_NUM, they will be reduce to a single INT_NUM node, like 100 + 100 becomes 200.

Code generator

The code generator will use the AST class to generate the code. Here are the ways of generate code:

1. Var declaration. For variable, they will all be stored in the frame. The space they need and their offset will be calculated in compile time and record in the symbol table. When they are use, the code generator can get the address from the symbol table.
2. Assignment. Get the address from the symbol table, then use lw to store the result to the address.
3. Control statement. Generate code for expression and code block, then generate code for condition jump.
4. Register allocation. Use the on the fly register allocation.



```
unsigned CG::getReg(AST* a1, AST* a2) {
    int res;
    // if one of them already in register, choose one of them
    if (a1 != nullptr && addrDespt.inReg(a1)) {
        return addrDespt.RegPos(a1);
    }
    else if (a2 != nullptr && addrDespt.inReg(a2)) {
        return addrDespt.RegPos(a2);
    }
    // try to get a free register
    else if ((res = regDespt.getFree()) != -1) {
        return res;
    }
    // swap out a filled register
    else {
        AST* node;
        res = regDespt.swap(node);
        addrDespt.deleteReg(node);
        spTop += 4;
        addrDespt.setStOffset(node, spTop);
        // mips code for store the register to stack
        cout << "addi $sp, $sp, -4" << endl;
        cout << "sw $" <<  res << ", ($sp)" << endl;

        return res;
    }
}
```

The code for getting a free register