Lê Minh Khôi - 1952076
Lương Duy Hưng - 1952747
Huỳnh Minh Trí - 1953041

**BKOOL.g4**

```
exp:
        LB exp RB
        | exp (MULOP | DIVOP) exp
        | exp (ADDOP | SUBOP) exp
        | funcall
        | INTLIT
        | FLOATLIT;

ADDOP: '+';

SUBOP: '-';

MULOP: '*';

DIVOP: '/';
```

**AST.py**

```
class BinaryOp(Expr):
    # op: str
    # left: Expr
    # right: Expr
    def __init__(self, op, left, right):
        self.op = op
        self.left = left
        self.right = right

    def __str__(self):
        return "BinaryOp(" + self.op + "," + str(self.left) + "," + str(self.right) + ")"

    def accept(self, v, param):
        return v.visitBinaryOp(self, param)
```

**Visitor.py**

```
@abstractmethod
    def visitBinaryOp(self, ast, param):
        pass
```

```
def visitBinaryOp(self, ast, param):
    return None
```

**ASTGeneration.py**

```
def visitExp(self, ctx: BKOOLParser.ExpContext):
    if (ctx.funcall()):
        return self.visit(ctx.funcall())
    elif ctx.INTLIT():
        return IntLiteral(int(ctx.INTLIT().getText()))
    elif ctx.FLOATLIT():
        return FloatLiteral(float(ctx.FLOATLIT().getText()))
    elif ctx.LB():
        return ctx.exp().accept(self)
    else:
        op = ctx.getChild(1).getText()
        left = ctx.exp(0).accept(self)
        right = ctx.exp(1).accept(self)
        return BinaryOp(op, left, right)
```

**CodeGenerator.py**

```
def visitBinaryOp(self, ast, o):
    #ast: BinarOp
    #o: Any

    ctxt = o
    frame = ctxt.frame
    left, left_type = ast.left.accept(self, o)
    right, _ = ast.right.accept(self, o)
    if ast.op in ["+", "-"]:
        return self.emit.emitADDOP(ast.op, left_type, frame), left_type
    elif ast.op in ["*", "/"]:
        return self.emit.emitMULOP(ast.op, left_type, frame), left_type
```

**CodeGenSuite.py**

```
def test_bin_ast_1(self):
    input = """void main() {putFloatLn(10 + 1);}"""
    expect = "10\n"
    self.assertTrue(TestAST.test(input, expect, 503))
```

Phạm Bùi Minh Huân 1952056
Ho Trí Kháng 1952069
Lâm Minh Quân 1952114

**BKOOL.G4**

```
exp: operand (ADD | SUB | MUL | DIV) operand | operand;
operand: funcall | INTLIT | FLOATLIT;
ADD: '+';
SUB: '-';
MUL: '*';
DIV: '/';
```

**AST.py**

```
class BinOp(Expr):
    #exp1: Expr
    #exp2: Expr
    #op: str
    def __init__(self, exp1, exp2, op):
        self.exp1 = exp1
        self.exp2 = exp2
        self.op = op

    def __str__(self):
        return "BinOp(" + str(op) + "," + str(self.exp1) + "," + str(self.exp2) + ")"

    def accept(self, v, param):
        return v.visitBinOp(self, param)
```

**ASTGeneration.py**

```
    def visitExp(self,ctx:BKOOLParser.ExpContext):
        if ctx.getChildCount() == 1:
            return self.visit(ctx.operand())
        else:
            exp1 = self.visit(ctx.operand(0))
            exp2 = self.visit(ctx.operand(1))
            op = ctx.getChild(1).getText()
            return BinOp(exp1, exp2, op)

    def visitOperand(self, ctx: BKOOLParser.OperandContext):
        if (ctx.funcall()):
            return self.visit(ctx.funcall())
        elif ctx.INTLIT():
            return IntLiteral(int(ctx.INTLIT().getText()))
```

```
        else:
            return FloatLiteral(float(ctx.FLOATLIT().getText()))
```

**CodeGenerator.py**

```
    def visitBinOp(self, ast , o):
        ctxt = o
        frame = ctxt.frame
        # assume lhs and rhs always have the same type
        if type(ast.exp1) is IntType:
            self.emit.printout(visitIntLiteral(self, ast.exp1, o))
        elif type(ast.exp1) is FloatType:
            self.emit.printout(visitFloatLiteral(self, ast.exp1, o))

        if type(ast.exp2) is IntType:
            self.emit.printout(visitIntLiteral(self, ast.exp2, o))
        elif type(ast.exp2) is FloatType:
            self.emit.printout(visitFloatLiteral(self, ast.exp2, o))

        if ast.op == '+' or ast.op == '-':
            self.emit.printout(self.emit.ADDOP(ast.op, ast.exp1, o))
        elif ast.op == '*' or ast.op == '/':
            self.emit.printout(self.emit.MULOP(ast.op, ast.exp1, o))
```

Nguyễn Minh Tâm - 1952968
Mai Tôn Nhật Khánh - 1852038
Nguyễn Luật Gia Khôi - 1952079

**BKOOL.g4**:

```
exp: funcall | INTLIT | FLOATLIT;
funcall: ID LB (exp | binop)? RB ;
binop: exp (PLUS | MINUS | MUL | DIV) exp;
PLUS: '+';
MINUS: '-';
MUL: '*';
DIV: '/';
```

**AST.py**:

```
class BinOp(Expr):
    #op1:Expr
    #op2:Expr
    #operator:str
    def __init__(self, operator, op1, op2):
        self.operator = operator
        self.op1 = op1
        self.op2 = op2

    def __str__(self):
        return "BinOp(" + str(self.operator) + ',' + str(self.op1) + ',' + str(self.op2) + ")"

    def accept(self, v, param):
        return v.visitBinOp(self, param)
```

**ASTGeneration.py**:

```
class ASTGeneration(BKOOLVisitor):
    def visitBinOp(self,ctx:BKOOLParser.BinopContext):
        return BinOp(ctx.getChild(1).getText(), self.visit(ctx.exp(0), self.visit(ctx.exp(1)))
```

Nguyen Le Nhat Duong 1952638
Trần Quốc Việt - 1953097
Huỳnh Phước Thiện - 1952463

**Question 2b.**
**BKOOL.g4:**

```
exp: exp ('+' | '-') exp1 | exp1;

exp1: exp1 ('*' | '/') exp2 | exp2;

exp2: funcall | INTLIT | FLOATLIT;
```

**ASTGeneration.py:**

```
    def visitExp(self,ctx:BKOOLParser.ExpContext):
        if ctx.getChildCount() == 3:
            e1 = self.visit(ctx.exp())
            e2 = self.visit(ctx.exp1())
            op = ctx.getChild(1).getText()
            return BinOP(op, e1, e2)
        else:
            return self.visit(ctx.exp1())
    def visitExp1(self,ctx:BKOOLParser.Exp1Context):
        if ctx.getChildCount() == 3:
            e1 = self.visit(ctx.exp1())
            e2 = self.visit(ctx.exp2())
            op = ctx.getChild(1).getText()
            return BinOP(op, e1, e2)
        else:
            return self.visit(ctx.exp2())
    def visitExp2(self,ctx:BKOOLParser.Exp2Context):
        if (ctx.funcall()):
            return self.visit(ctx.funcall())
        elif(ctx.FLOATLIT()):
            return FloatLiteral(float(ctx.FLOATLIT().getText()))
        else:
            return IntLiteral(int(ctx.INTLIT().getText()))
```

**Visitor.py:**

**CodeGenerator.py:**

```
    def visitBinOP(self, ast, o):
        #ast: BinOP
        #o: Any
        ctxt = o
        frame = ctxt.frame
```

```
        frame.push()
        frame.push()
        myType = FloatType if ast.e1 is FloatType or ast.e2 is FloatType
else IntType
        return self.emit.emitADDOP(ast.op, myType, frame), myType
```

**AST.py:**

```
class BinOP(Expr):
    def __init__(self, op, e1, e2):
        self.op = op
        self.e1 = e1
        self.e2 = e2

    def accept(self, v, param):
        return v.visitBinOP(self, param)

    def __str__(self):
        return "BinOp(" + self.op + ", " + str(self.e1) + ", " +
str(self.e2) + ")"
```

**CodeGenSuite.py**

```
    def test_binOp_1(self):
        input = Program([
            FuncDecl(Id("main"),[],VoidType(),Block([],[
                BinOP('+',2,3)]))])
        expect = "5.9"
        self.assertTrue(TestCodeGen.test(input,expect,503))
```

Le Duc Cam - 1952588
Quach Dang Giang: 1952044
BKOOL.g4

```
exp: funcall | INTLIT | FLOATLIT | bin_exp;
bin_exp: (INTLIT | FLOATLIT) ( ADD | MINUS | MUL  | DIV ) (INTLIT |
FLOATLIT) ;
ADD:'+';
```

```
MINUS:'-';
MUL:'*';
DIV:'/';
```

AST.py

```
class BinExpr(Expr):
    def __init__(self, op, left, right):
        self.op = op
        self.left = left
        self.right = right
    def __str__(self):
        return "BinExpr(" + str(self.op) + ',' + str(self.left) + ',' +
str(self.right) + ')'
    def accept(self, v, param):
        return v.visitBinExpr(self, param)
```

Visitor.py

```
class Visitor(ABC):
     @abstractmethod
    def visitBinExpr(self, ast, param):
        pass

class BaseVisitor(Visitor):
     def visitBinExpr(self, ast, param):
        return None
```

ASTGeneration.py

```
def visitExp(self,ctx:BKOOLParser.ExpContext):
        if (ctx.funcall()):
            return self.visit(ctx.funcall())
```

```
        elif (ctx.bin_exp()):
            return self.visit(ctx.bin_exp())
        elif ctx.FLOATLIT():
            return FloatLiteral(float(ctx.FLOATLIT().getText()))
        else:
            return IntLiteral(int(ctx.INTLIT().getText()))
```

```
    def visitExp(self, ctx: BKOOLParser.ExpContext):
        if (ctx.funcall()):
            return self.visit(ctx.funcall())
        elif ctx.INTLIT():
            return IntLiteral(int(ctx.INTLIT().getText()))
        elif ctx.FLOATLIT():
            return FloatLiteral(float(ctx.FLOATLIT().getText()))
        elif ctx.LB():
            return ctx.exp().accept(self)
        else:
            op = ctx.getChild(1).getText()
            left = ctx.exp(0).accept(self)
            right = ctx.exp(1).accept(self)
            return BinaryOp(op, left, right)
```

CodeGenSuite.py

```
def test_bin(self):
        """Simple program: int main() {} """
        input = """void main() {putBin(100+200);}"""
        expect = "300"
        self.assertTrue(TestCodeGen.test(input,expect,504))
```

—----------

Đỗ Đăng Khoa - 1952295
Phạm Nhựt Huy - 1952059
Trần Anh Thái - 1752494

BKOOL.g4

exp: operand OP operand | rest;
rest: funcall | operand; //
operand: INTLIT | FLOATLIT;
OP: [+-*/];

AST.py

```
class BinExp(Expr):
    def __init__(self, oprt, oprd1, oprd2):
        self.oprt = oprt #str
        self.oprd1 = oprd1 #Expr
        self.oprd2 = oprd2 #Expr

    def __str__(self):
        return "BinExpr(" + self.oprt + ", " + str(self.oprd1) + ", " + str(self.oprd2) + ")"

    def accept(self, v, param):
        return v.acceptBinExpr(self, param)
```

Visitor.py

```
@abstractmethod
def visitBinExpr(self, ast, param):
            pass

def visitBinExpr(self, ast, param):
        return None
```

ASTGenerator.py

```
def visitExp(self,ctx:BKOOLParser.ExpContext): #
        if (ctx.OP()):
            o1 = self.visit(ctx.oprd1())
            o2 = self.visit(ctx.oprd2())
            return BinExpr(ctx.OP().getText(), o1, o2)
        else:
            return self.visit(ctx.rest())

def visitRest(self, ctx:BKOOLParser.RestContext):
        if (ctx.funcall()):
            return self.visit(ctx.funcall())
        else:
            return self.visit(ctx.operand())

def visitOperand(self, ctx:BKOOLParser.OperandContext):
        if ctx.INTLIT():
            return IntLiteral(int(ctx.INTLIT().getText()))
        else:
            return FloatLiteral(float(ctx.FLOATLIT().getText()))
```

CodeGenerator.py

```
def visitFloatLiteral(self, ast, o):
```

```
        #ast: FloatLiteral
        #o: Any

         ctxt = o
        frame = ctxt.frame
        return self.emit.emitPUSHFCONST(ast.value, frame), FloatType()

def visitBinExpr(self, ast, o):
        #oprt, oprd1, oprd2

        ctxt = o
        frame = ctxt.frame
        (e1, t1) = self.visit(ast.oprd1,o)
        (e2, t2) = self.visit(ast.oprd2,o)
        typee = IntType()
        if type(t1) == FloatType or type(t2) == FloatType:
            typee = FloatType()
        if ast.oprt == "+":
            return e1 + e2 + self.emit.emitADDOP("+", typee, frame), typee
        elif ast.oprt == "-":
            return self.emit.emitADDOP("-", typee, frame)
        elif ast.oprt == "*":
            return self.emit.emitMULOP("*", typee, frame)
        elif ast.oprt == "/":
            return self.emit.emitMULOP("/", typee, frame)
```

==========

Phạm Khánh Trình - 1953044
Võ Ngọc Sang - 1952430
Cao Bá Huy - 1952713

**a) BKOOL.g4**

```
exp: exp ADDOP | SUBOP exp1
           | exp1;

exp1: exp1 MULOP | DIVOP exp2
           | exp2;

exp2: funcall | INTLIT | FLOATLIT;
```

**AST.py**

```
class BinaryOp(Expr):
```

```
op: str
left: Expr
right: Expr

def __str__(self):
    return "BinaryOp(" + self.op + "," + str(self.left) + "," + str(self.right) + ")"
def accept(self, v, param):
    return v.visitBinaryOp(self, param)
```

**ASTGeneration.py**

```
def visitExp(self,ctx:BKOOLParser.ExpContext):
    if ctx.exp():
        op = ctx.ADDOP().getText() if ctx.ADDOP() else ctx.SUBOP().getText()
        left = self.visit(ctx.exp())
        right = self.visit(ctx.exp1())

        return BinaryOp(op,left,right)
    else:
        return self.visit(ctx.exp1())

def visitExp1(self,ctx:BKOOLParser.Exp1Context):
    if ctx.exp1():
        op = ctx.MULOP().getText() if ctx.MULOP() else ctx.DIVOP().getText()
        left = self.visit(ctx.exp1())
        right = self.visit(ctx.exp2())

        return BinaryOp(op,left,right)
    else:
        return self.visit(ctx.exp2())

def visitExp2(self,ctx:BKOOLParser.Exp2Context):
    if ctx.funcall():
        return self.visit(ctx.funcall())
    elif ctx.INTLIT():
        return ctx.INTLIT().getText()
    else:
        return ctx.FLOATLIT().getText()
```

**CodeGenerator.py**

```
def visitBinaryOp(self, ast, o):
    ctxt = o
    frame = ctxt.frame
    if ast.op == "+":
```

```
        left = self.emit.emitREADVAR(str(self.visit(ast.left)),type(self.vist(ast.left)),0,frame)
        right = self.emit.emitREADVAR(str(self.visit(ast.right)),type(self.vist(ast.right)),0,frame)
        return self.emit.emitADDOP(ast.op,type(left),frame)
    elif ast.op == "*":
        left = self.emit.emitREADVAR(str(self.visit(ast.left)),type(self.vist(ast.left)),0,frame)
        right = self.emit.emitREADVAR(str(self.visit(ast.right)),type(self.vist(ast.right)),0,frame)
        return self.emit.emitMULOP(ast.op,type(left),frame)
    else:
        return self.emit.emitDIV(frame)
```