Compiler Project

Bhumik Shah – 201201055

Steps to run the code

1. Run the make file.
    >> make clean
    >> make

2. Run the executable giving it a decaf program as input.
>> ./astbuilder <decaf program>

Work We Did:

1. In bison.ypp file we have written grammer using tokens specified in flex.l file.

2. To build AST. We decided what kind of grammer language should accept that's the syntax and using that decided heirarchy of various classes.

3. We used print() function of AST defined in it's classes.

4. To write LLVM, we have to include all header files and similiar to print we had to make Codegen function for each class.

5. The LLVM is finally generated using Codegen function of each class the IRBuilder of LLVM knows where to insert which instruction.

Reference :
http://llvm.org/docs/tutorial/index.html

Problems we faced

1) Shift Reduce Errors

Errors We Got

>> In bison.output file there are all shift reduce errors that we obtained and the entire state diagrams corresponding to our grammer.

```
 State 95
   64 EXPR: EXPR . BIN_OP EXPR
   66     | NOT EXPR .
    MODULUS  shift, and go to state 97
    MODULUS   [reduce using rule 66 (EXPR)]
    $default  reduce using rule 66 (EXPR)
    BIN_OP    go to state 110
    ARITH_OP  go to state 111
    REL_OP    go to state 112
    EQ_OP     go to state 113
    COND_OP   go to state 114
State 96 : ARITH_OP: MODULUS .
```

Reason for the errors is grammer we used

```
Expr: EXPR BIN_OP EXPR
BIN_OP: ARITH_OP
| REL_OP
| EQ_OP
| COND_OP
;
ARITH_OP: PLUS
| MINUS
| MULTIPLY
| DIVIDE
| MODULUS
```

Solved the issue by changing the grammer and adding precedence rule for the operators

Precedence rule added :-

```
%left TOK_OR
%left TOK_AND
%left TOK_EQUAL_EQUAL TOK_NOT_EQUAL
%nonassoc TOK_LESS TOK_GREATER TOK_LESS_EQUAL TOK_GREATER_EQUAL
%left TOK_PLUS TOK_MINUS
%left TOK_MULTIPLY TOK_DIVIDE TOK_MODULUS
```

Grammer Changed : -

```
EXPR: LOCATION {$$=$1;}
    | METHOD_CALL {$$=$1;}
| LITERAL {$$=$1;}
| EXPR TOK_PLUS EXPR  { $$ = new BinaryExpr(ADD,$1,$3); }
| EXPR TOK_MINUS EXPR  { $$ = new BinaryExpr(SUB,$1,$3); }
| EXPR TOK_MULTIPLY EXPR  { $$ = new BinaryExpr(MUL,$1,$3); }
| EXPR TOK_DIVIDE EXPR  { $$ = new BinaryExpr(DIV,$1,$3); }
| EXPR TOK_MODULUS EXPR  { $$ = new BinaryExpr(MOD,$1,$3); }
| EXPR TOK_GREATER EXPR  { $$ = new BinaryExpr(GT,$1,$3); }
| EXPR TOK_GREATER_EQUAL EXPR  { $$ = new BinaryExpr(GEQ,$1,$3); }
| EXPR TOK_LESS EXPR   { $$ = new BinaryExpr(LT,$1,$3); }
| EXPR TOK_LESS_EQUAL EXPR  { $$ = new BinaryExpr(LEQ,$1,$3); }
```

2) Errors In Heirarchy Of Classes

Initially we were not able to capture the Hierarchical structure of the program. Failed to realized pointer to Parent Class can also point to Child Class so we could store all Instance of different Child Classes using array of Parent classes

```
IfStmt
ForStmt
ExprStmt
BlockStmt
LocationStmt
BreakStmt
ReturnStmt
ContinueStmt
```

Could be captured together by defining

```
list <Stmt *> arr;
```

3) Did'nt know what did virtual mean and it's uses

For 2nd phase of our project we were able to capture entire Hierarchical structure of our initial program but for llvm phase we needed each child node to have different Codegen function so we had to use “virtual” functions in parent which were redefine in child classes.Initially we were unaware of this so we got errors

```
project:151: undefined reference to `vtable for AstNode'
project:151: undefined reference to `vtable for AstNode'
collect2: ld returned 1 exit status
make: *** [AstNode] Error 1
```

4) LLVM CodeGen Errors

As mentioned earlier that a pointer to Parent can point to the Child but if we want some specific method only in child and we want to access that method from pointer of parent we required to define a virtual function with same name and type in parent which essentially returns arbitary hardcoded value.

```
eg.
class GlobalDeclStmt:public Stmt
{
public:
virtual Value *Codegen()
{
  list<AstNode*>::iterator i;
  i=var_list1.begin();
  (*i) > return_name();
  }
}
```

In this case list var_list1 was of type VariableChild but which is child of AstNode and since VariableChild has function return_name() but AstNode does not have it so we had to write a virtual function with same name and type in parent which essentially returns arbitary hardcoded value.

5) Undefined reference to Builder.store() :

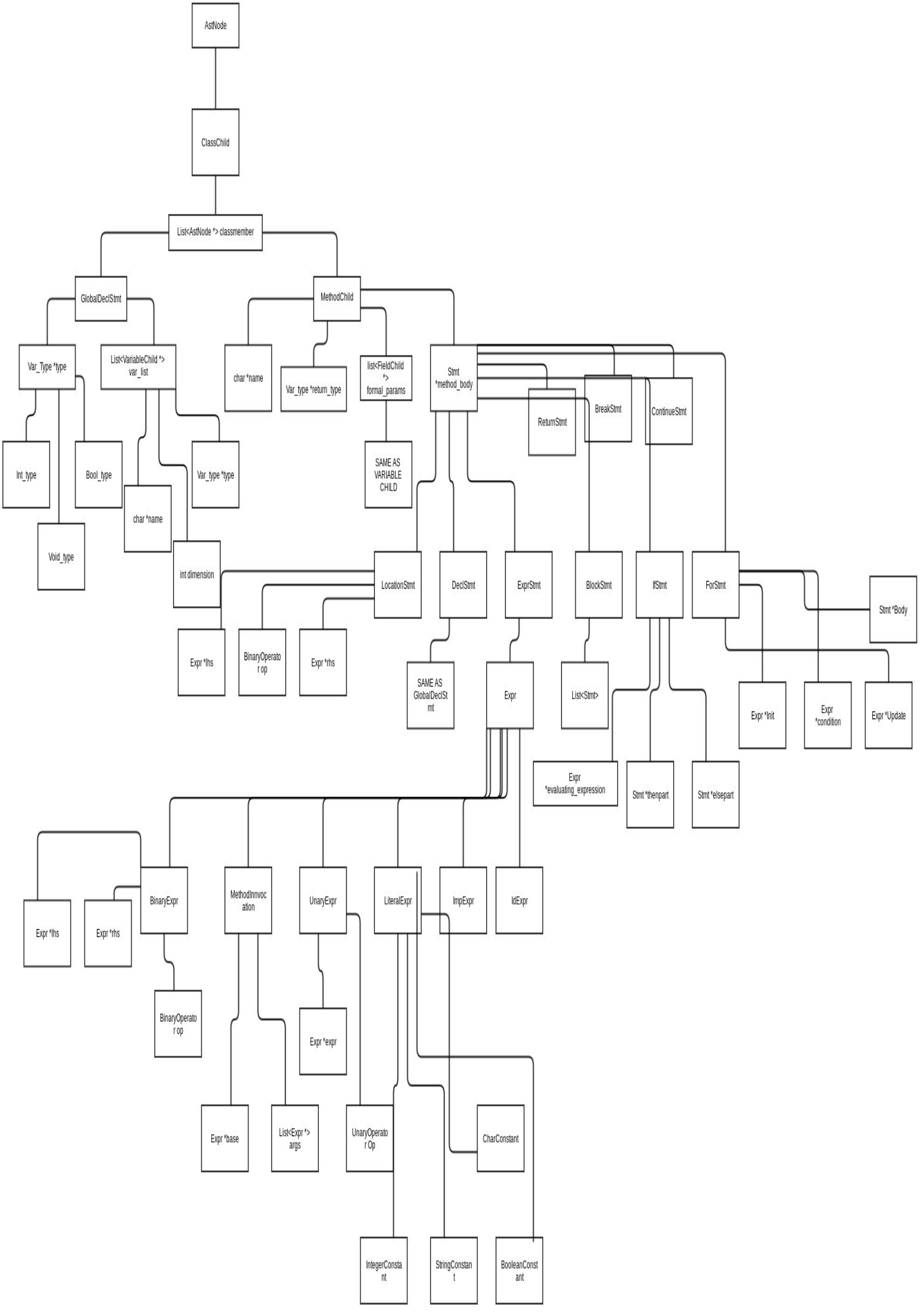
Make File had to be modified flags were put at the end.

6) Generating LLVM CodeGen for Break , Continue , Array Reference (accessing a particular index) could not be done

Will solve the issue by 10 th December

Updates

Solved break, continue, array reference issues but while solving break,continue issues we had to comment out function verify part which is very bad practise as it's important after creating llvm for a particular function that the function needs to be verified if something is wrong with the function or not so that's a problem and during first viva came to know about the issue that callout should be used to call standart c functions that was not done as way we implemented callout was we stored all the function that were defined in a module that module is something that defines all basic block so since standard C functions are not registered in the same way we cannot use callout for them.


AstNode
ClassChild
List<AstNode *> classmember
GlobalDeclStmt
MethodChild
Var_Type *type
List<VariableChild *> var_list
char *name
Var_type *return_type
list<FieldChild *> formal_params
Stmt *method_body
ReturnStmt
BreakStmt
ContinueStmt
Int_type
Bool_type
Void_type
char *name
int dimension
Var_type *type
SAME AS VARIABLE CHILD
LocationStmt
DeclStmt
ExprStmt
BlockStmt
IfStmt
ForStmt
Stmt *Body
Expr *lhs
BinaryOperator op
Expr *rhs
SAME AS GlobalDeclStmt
Expr
List<Stmt>
Expr *Init
Expr *condition
Expr *Update
Expr *evaluating_expression
Stmt *thenpart
Stmt *elsepart
Expr *lhs
Expr *rhs
BinaryExpr
MethodInnvocation
UnaryExpr
LiteralExpr
ImpExpr
IdExpr
BinaryOperator op
Expr *expr
Expr *base
List<Expr *> args
UnaryOperator Op
CharConstant
IntegerConstant
StringConstant
BooleanConstant