Compilers Project Report

Team

Name: Garima Ahuja
Roll No: 201002017

Name: Anupriya I
Roll No: 201203009

Bonus

By the middle of the project, we started enjoying and experimenting with the project and hence implemented semantics which were not required.

1. Scoping

```
phase-3 — bash — 114×33
flier@Anupriyas-MacBook-Air:~/Dropbox/Academics/Compilers/Bitbucket/phase-3$ cat g_7.decaf
//Scoping
class Program {
    int a;
    int main() {
        int a;
        a = 4;
        return a;
    }
    int func() {
        a = 5;
        return a;
    }
}
flier@Anupriyas-MacBook-Air:~/Dropbox/Academics/Compilers/Bitbucket/phase-3$ ./makefile g_7.decaf
; ModuleID = 'Look at the pretty IR'

@a = common global i32 5, align 4

define i32 @main() {
entry:
  %a = alloca i32
  store i32 0, i32* %a
  store i32 4, i32* %a
  %a1 = load i32* %a
  ret i32 %a1
}

define i32 @func() {
entry:
  ret i32 5
}
flier@Anupriyas-MacBook-Air:~/Dropbox/Academics/Compilers/Bitbucket/phase-3$
```

2. Return Value mismatch check

```
Terminal  Shell  Edit  View  Window  Help
phase-3 — bash — 114×33
flier@Anupriyas-MacBook-Air:~/Dropbox/Academics/Compilers/Bitbucket/phase-3$ cat g_1.decaf
class Program {
    //Return type
    int main() {
        return true;
    }
}
flier@Anupriyas-MacBook-Air:~/Dropbox/Academics/Compilers/Bitbucket/phase-3$ ./makefile g_1.decaf
Error: Return type doesn't match function prototype
; ModuleID = 'Look at the pretty IR'

define i32 @main() {
entry:
  ret i1 true
}
flier@Anupriyas-MacBook-Air:~/Dropbox/Academics/Compilers/Bitbucket/phase-3$
```

Dilemmas faced while building the Compiler

1. Compiling the Compiler

Understanding how Flex and Bison work together helped in writing simple makefile. All the makefiles that we found on internet had hundreds of lines of code and were not at all understandable.

2. Phi Node

Making Phi Nodes was difficult because the llvm documentation doesn't provide function descriptions. We had to do a lot of guess work here. And the Phi node would take Value and Basicblock only, there was no llvm abstraction available that would directly take Block and BasicBlock and return you the corresponding Phi Node by checking what all variable conflicts. We had to do this by ourselves.

```
phase-3 — bash — 137×51

flier@Anupriyas-MacBook-Air:~/Dropbox/Academics/Compilers/Bitbucket/phase-3$ cat g_8.decaf
//if and phi
class Program {
    int main() {
        int b;
        int a;
        if(true) {
            b = b+1;
            a=5;
        } else {
            b=b+2;
            a=7;
        }
        b=a+1;
    }
}
flier@Anupriyas-MacBook-Air:~/Dropbox/Academics/Compilers/Bitbucket/phase-3$ ./makefile g_8.decaf
; ModuleID = 'Look at the pretty IR'

define i32 @main() {
entry:
  %a = alloca i32
  %b = alloca i32
  store i32 0, i32* %b
  store i32 0, i32* %a
  br i1 true, label %then, label %else

then:                                             ; preds = %entry
  %b1 = load i32* %b
  %add = add i32 %b1, 1
  store i32 %add, i32* %b
  store i32 5, i32* %a
  br label %ifcont

else:                                             ; preds = %entry
  %b2 = load i32* %b
  %add3 = add i32 %b2, 2
  store i32 %add3, i32* %b
  store i32 7, i32* %a
  br label %ifcont

ifcont:                                           ; preds = %else, %then
  %a4 = phi i32 [ 5, %then ], [ 7, %else ]
  %b5 = phi i32 [ %add, %then ], [ %add3, %else ]
  %a6 = load i32* %a
  %add7 = add i32 %a6, 1
  store i32 %add7, i32* %b
}
flier@Anupriyas-MacBook-Air:~/Dropbox/Academics/Compilers/Bitbucket/phase-3$
```

3. Array Access

Variables were easy, Arrays were not because of not enough resources.

```
Terminal  Shell  Edit  View  Window  Help
phase-3 — bash — 204×51
flier@Anupriyas-MacBook-Air:~/Dropbox/Academics/Compilers/Bitbucket/phase-3$ cat dotProduct.decaf
class Program {
    int a[10], b[10];
        int dot_product()
        {
                int result;
                result = 0;
                int i;
                for i=0 , i<10 {
                        result+=a[i]*b[i];
                }
                return result;
        }

}
flier@Anupriyas-MacBook-Air:~/Dropbox/Academics/Compilers/Bitbucket/phase-3$ ./makefile dotProduct.decaf
; ModuleID = 'Look at the pretty IR'

@a = common global [10 x i32], align 16
@b = common global [10 x i32], align 16

define i32 @dot_product() {
entry:
  %i = alloca i32
  %result = alloca i32
  store i32 0, i32* %result
  store i32 0, i32* %result
  store i32 0, i32* %i
  br label %loop

loop:                                             ; preds = %loop, %entry
  store i32 0, i32* %i
  %result1 = load i32* %result
  %0 = load i32* getelementptr inbounds ([10 x i32]* @a, i64 0, i64 0)
  %1 = load i32* getelementptr inbounds ([10 x i32]* @b, i64 0, i64 0)
  %mul = mul i32 %0, %1
  %add = add i32 %result1, %mul
  store i32 %add, i32* %result
  %i2 = load i32* %i
  %"less than" = icmp ult i32 %i2, 10
  %i3 = load i32* %i
  %nextvar = add i32 %i3, 1
  store i32 %nextvar, i32* %i
  %loopcond = icmp ne i1 %"less than", true
  br i1 %loopcond, label %loop, label %afterloop

afterloop:                                        ; preds = %loop
  %result4 = load i32* %result
  ret i32 %result4
}
flier@Anupriyas-MacBook-Air:~/Dropbox/Academics/Compilers/Bitbucket/phase-3$
```

4. llvm Documentation

As function descriptions are not present in the llvm documentation, we had to look at the arguments and guess what some particular function did. This was very time consuming.

CLASS Hierarchy

class Program

class Decl

- **class FieldDecl : public Decl**
- **class MethodDecl : public Decl**

class Expr

- **class UnaryExpr : public Expr**

```
        class BinaryExpr : public Expr
        class Location : public Expr
        class MethodCall : public Expr
        class Constants : public Expr

class Statement
        class MethodCallStatement
        class BlockStatement
        class FieldDeclStatement
        class IfStatement
        class ForStatement
        class KeywordStatement
        class AssignmentStatement

class Param
```