Mini Assignment #1
An Introduction to the LLVM Infrastructure, AST, IR and Compiler Options

Puneet Mangla (CS17BTECH11029)

Download LLVM :

```
$ git clone https://github.com/llvm/llvm-project.git

$ cd llvm-project

$ mkdir build

$ cd build

$ cmake -G Unix Makefiles -DCMAKE_BUILD_TYPE=Release
-DLLVM_ENABLE_ASSERTIONS=On ../llvm
$ sudo make -j2
$ sudo make install
$ sudo apt-get install clang
```

Directory Layout:

1. llvm/examples : contains examples using llvm-IR and JIT.
2. llvm/include : header files/sub-directories for different parts of llvm like Analysis, CodeGen, Target etc. Generic support libraries like C++ STL utilities.
3. llvm/lib : It contains the source code files for llvm-IR, Assembly-parser, CodeGen, Target etc.
4. llvm/projects : directory which you can use to create your own llvm based projects. Empty for fresh-build.
5. Test-suite : contains comprehensive test-suites for evaluating correctness and performance.
6. llvm/tools : executable tools built upon library above. Provides good user interface and includes assembler, disassembler, linker etc.
7. llvm/utils : utilities for working with llvm-source code. Eg : vim syntax highlighter for LLVM assembly files and TableGen description file.

Clang AST Structure:

1. The clang AST resembles C++ code and C++ standard which makes it a good fit for refactoring tools.
2. Enable AST dump mode by using flag -ast-dump
3. clang -Xclang -ast-dump -fsyntax-only test.cc
4. The top level declaration unit is transition unit declaration (TranslationUnitDecl). Every AST start from this declaration followed by user declaration.
5. Clang AST has class hierarchy and so there is no formulation of common ancestor.
6. Most AST nodes derive form large basic nodes like Decl and Stmt.
7. Every node contains the token type , memory address, line number,lvalue etc.

8. Clang automatically applies type casting wherever necessary. Some common casting are FunctionToPointerDecay, BitCast, ArrayToPointerDecay, FloatingCast, LValueToRValue etc.
9. The clang AST contains a common AST part corresponding to header library like <stdio.h>
10. Some common AST nodes are CompoundStmt, DeclStmt, VarDecl, CallExpr, ImplicitCastExpr, DeclRefExpr, ImplicitCastExpr, StringLiteral, IfStmt etc.

Clang AST Traversal:

1. RecursiveASTVisitor is a class that does pre/post order DFS of a clang AST and visits each node.
2. #include "clang/AST/RecursiveASTVisitor.h"
3. We can walk up in the hierarchy from an AST node until we reach the topmost node.
4. We can define and call user-overridable functions to actually visit a node.
5. TraverseDecl(Decl *x) is the entry point for traversing an AST rooted at x. It dispatches to many dynamic type functions and visits child of node x. TraverseStmt(Stmt *x) and TraverseType(QualType x) work similarly.
6. WalkUpFromFoo(Foo *x) doesn't visit the child of x instead it call a functions that jumps to its direct parent and visit it.
7. One can override Traverse*, WalkFrom*, Visit* methods for declarations, types, statements, expressions, or other AST nodes to achieve customization behavior.
8. Returning false from one of these overridden functions will abort the entire traversal.
9. By default traversal order is pre-order.

Error-Messages:

1. Use assertions to check pre/post conditions which help to find bugs early and improves debugging. Program behaviour is undefined If an assertion evaluates to false.
2. The "<cassert>" header file is probably already included by the header files you are using, so it doesn't cost anything to use it.
3. Eg : assert(idx < getNumSuccessors() && "Successor # out of range!");
4. Earlier assets were used to mark pieces of code that should not be reached.
5. Now have: llvm_unreachable:
6. llvm_unreachable("Invalid radix for integer literal") which will print the message if it's ever reached and then exit the program.

LLVM-IR

1. llvm-IR allows llm to parse multiple source languages and generate their corresponding codes. The IR is the point where the majority of machine independent optimizations take place.
2. **clang sum.c -emit-llvm -S -c -o sum.ll**
3. The entire LLVM IR file defines a module which is the top-most data structure in a '.ll' file. The module further consists of a sequence of functions, that contains sequence of basic block and instructions.

4. Llvm local values are analogous to registers in assembly language. They start with a % sign followed by name of variable. Eg: *%add = add nsw i32 %0, %1* will add local value *%0* to *%1* and put the result in the new local value, *%add*.
5. IT uses single static assignment (SSA) where every value is assigned once. This simplicity allows space for more optimizations and back-tracing.
6. It has infinite number of local values (starts with %).
7. The function declarations almost matches with C . i32 represent a 32 bit integer. Local variables se % prefix whereas global use @.
8. Function body is divided into basic block. Each basic block starts with a label. A label to a basic block is like an identifier to an instruction.
9. Each basic block should end with a return instruction which takes it to another label. Each function has a special label 'entry'. No label should target entry label after termination.
10. Alloca reserves space in the stack frame of the function. The space amount is defined by type identifier like i32 , i64 etc.
11. %a local variables are stored in stack and their address is denoted by %a.addr.

Assembly Language:

1. Languages that don't allow function overloading mostly don't require name mangling.
2. In c++, one may define two functions with the same name and different signature known as function overloading. In this case the type information will be encoded in function name: Eg:
   int f (void) { return 1; }
   int f (int) { return 0; }
   Resolves to :
   int __f_v (void) { return 1; }
   int __f_i (int) { return 0; }

3. All mangled symbols begin with **_Z** . For nested names, this is followed by **N**, then a series of <length, id> pairs and finally **E**. For functions, this is then followed by the type information.
4. For example, suppose class X has a function: *int add(double a, short b),* its mangled name is *_ZN1X3addEds*, and a subclass Y overrides that function: *int add(float a, int b)*, whose mangled name is: *_ZN1Y3addEfi*. Another example of name mangling, when plus (+) operator is overloaded for class X, is: *_ZN1XplERKS.*

Compiler toolchain and options:

1. Lli and O3/O2 optimization- It is llvm interpreter that directly interprets LLVM bit code. It functions as a JIT compiler and execute the code *much* faster than the interpreter. Using bubble sort for sorting array of 50000 elements took:

   ```
   $ clang bsort.c -o bsort
   $ time ./bsort
   ```

6 seconds

```
$ clang -O3 -emit-llvm bsort.c -c -o bsort.bc
$ time lli bsort.bc
```

0.83 seconds

2. Llvm-dis : It is an diassembler that converts llvm-bitcode directly to assembly language.
3. Llc : Compile the program to native assembly code. Which can be further used by gcc to convert assembly code to program.

```
$ llc bsort.bc -o bsort.s
$ gcc bsort.s -o bsort.native
$ ./bsort.native
```

4. Opt : opt reads a llvm-bitcode and passed it to various llvm transformations to generate another bit code : Eg opt [options] [filename].
5. -{passname}: run any of LLVM's optimization or analysis passes in any order. The order in which the options occur on the command line are the order in which they are executed.
6. -time-passes : Record the amount of time needed for each pass and print it to standard error.

Kaleidoscope:

1. Only datatype in Kaleidoscope is a 64-bit floating point type.
2. We can use extern keyword to define library functions to be used later.
3. It support if/then/else, for loops, user defined data-types.
4. For AST, Kaleidoscope have expressions, a prototype, and a function object.
5. After AST we move on to IR. In order to generate LLVM IR, wefirst define virtual code generation (VisitChildren) methods in each AST class.
6. The builder object makes it easy to generate LLVM instructions. It keeps track of the current place to insert instructions and has methods to create new instructions.
7. module is an LLVM construct that contains functions and global variables. It will own the memory for all of the IR that we generate.
8. After IR generation, the IR will be sent through llvm which will apply machine independent optimizations.
9. LLVM makes it very easy to add JIT/Code generation support once the frontend has done its work (Generating the AST).

References:

1. https://llvm.org/docs/GettingStarted.html
2. https://llvm.org/docs/CodingStandards.html#assert-liberally
3. https://clang.llvm.org/doxygen/classclang_1_1RecursiveASTVisitor.html
4. https://hub.packtpub.com/introducing-llvm-intermediate-representation/
5. https://en.wikipedia.org/wiki/Name_mangling

Appendix : LLVM-IR Files of 5 non-trivial programs

1. Prime number:

```
; ModuleID = 'prime.c'
source_filename = "prime.c"
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-pc-linux-gnu"

@.str = private unnamed_addr constant [3 x i8] c"%u\00", align 1
@.str.1 = private unnamed_addr constant [17 x i8] c"Number is prime\0A\00", align 1
@.str.2 = private unnamed_addr constant [21 x i8] c"Number is composite\0A\00", align 1

; Function Attrs: noinline nounwind optnone uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4
  %4 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  %5 = call i32 (i8*, ...) @__isoc99_scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i32 0, i32 0),
i32* %2)
  store i32 1, i32* %4, align 4
  store i32 2, i32* %3, align 4
  br label %6

; <label>:6:                                ; preds = %19, %0
  %7 = load i32, i32* %3, align 4
  %8 = load i32, i32* %3, align 4
  %9 = mul nsw i32 %7, %8
  %10 = load i32, i32* %2, align 4
  %11 = icmp ule i32 %9, %10
  br i1 %11, label %12, label %22

; <label>:12:                               ; preds = %6
  %13 = load i32, i32* %2, align 4
  %14 = load i32, i32* %3, align 4
  %15 = urem i32 %13, %14
  %16 = icmp eq i32 %15, 0
  br i1 %16, label %17, label %18

; <label>:17:                               ; preds = %12
  store i32 0, i32* %4, align 4
  br label %22

; <label>:18:                               ; preds = %12
  br label %19

; <label>:19:                               ; preds = %18
  %20 = load i32, i32* %3, align 4
```

```
  %21 = add nsw i32 %20, 1
  store i32 %21, i32* %3, align 4
  br label %6

; <label>:22:                                     ; preds = %17, %6
  %23 = load i32, i32* %4, align 4
  %24 = icmp eq i32 %23, 1
  br i1 %24, label %25, label %27

; <label>:25:                                     ; preds = %22
  %26 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([17 x i8], [17 x i8]* @.str.1, i32 0, i32 0))
  br label %29

; <label>:27:                                     ; preds = %22
  %28 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([21 x i8], [21 x i8]* @.str.2, i32 0, i32 0))
  br label %29

; <label>:29:                                     ; preds = %27, %25
  ret i32 0
}

declare i32 @__isoc99_scanf(i8*, ...) #1

declare i32 @printf(i8*, ...) #1

attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-divide-sqrt-fp-math"="false"
"disable-tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-elim"="true"
"no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false"
"no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="false"
"stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+fxsr,+mmx,+sse,+sse2,+x87"
"unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable-tail-calls"="false"
"less-precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-non-leaf"
"no-infs-fp-math"="false" "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false"
"no-trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="x86-64"
"target-features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }

!llvm.module.flags = !{!0}
!llvm.ident = !{!1}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{!"clang version 6.0.0-1ubuntu2 (tags/RELEASE_600/final)"}
```

2. Bubble sort:

```
; ModuleID = 'bsort.c'
source_filename = "bsort.c"
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-pc-linux-gnu"
```

```
@.str = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@.str.1 = private unnamed_addr constant [4 x i8] c"%d \00", align 1
@.str.2 = private unnamed_addr constant [2 x i8] c"\0A\00", align 1

; Function Attrs: noinline nounwind optnone uwtable
define void @swap(i32*, i32*) #0 {
  %3 = alloca i32*, align 8
  %4 = alloca i32*, align 8
  %5 = alloca i32, align 4
  store i32* %0, i32** %3, align 8
  store i32* %1, i32** %4, align 8
  %6 = load i32*, i32** %3, align 8
  %7 = load i32, i32* %6, align 4
  store i32 %7, i32* %5, align 4
  %8 = load i32*, i32** %4, align 8
  %9 = load i32, i32* %8, align 4
  %10 = load i32*, i32** %3, align 8
  store i32 %9, i32* %10, align 4
  %11 = load i32, i32* %5, align 4
  %12 = load i32*, i32** %4, align 8
  store i32 %11, i32* %12, align 4
  ret void
}

; Function Attrs: noinline nounwind optnone uwtable
define void @sort(i32*, i32) #0 {
  %3 = alloca i32*, align 8
  %4 = alloca i32, align 4
  %5 = alloca i32, align 4
  %6 = alloca i32, align 4
  store i32* %0, i32** %3, align 8
  store i32 %1, i32* %4, align 4
  store i32 0, i32* %5, align 4
  br label %7

; <label>:7:                                ; preds = %48, %2
  %8 = load i32, i32* %5, align 4
  %9 = load i32, i32* %4, align 4
  %10 = sub nsw i32 %9, 1
  %11 = icmp slt i32 %8, %10
  br i1 %11, label %12, label %51

; <label>:12:                               ; preds = %7
  store i32 0, i32* %6, align 4
  br label %13

; <label>:13:                               ; preds = %44, %12
  %14 = load i32, i32* %6, align 4
  %15 = load i32, i32* %4, align 4
  %16 = load i32, i32* %5, align 4
```

```
  %17 = sub nsw i32 %15, %16
  %18 = sub nsw i32 %17, 1
  %19 = icmp slt i32 %14, %18
  br i1 %19, label %20, label %47

; <label>:20:                                     ; preds = %13
  %21 = load i32*, i32** %3, align 8
  %22 = load i32, i32* %6, align 4
  %23 = sext i32 %22 to i64
  %24 = getelementptr inbounds i32, i32* %21, i64 %23
  %25 = load i32, i32* %24, align 4
  %26 = load i32*, i32** %3, align 8
  %27 = load i32, i32* %6, align 4
  %28 = add nsw i32 %27, 1
  %29 = sext i32 %28 to i64
  %30 = getelementptr inbounds i32, i32* %26, i64 %29
  %31 = load i32, i32* %30, align 4
  %32 = icmp sgt i32 %25, %31
  br i1 %32, label %33, label %43

; <label>:33:                                     ; preds = %20
  %34 = load i32*, i32** %3, align 8
  %35 = load i32, i32* %6, align 4
  %36 = sext i32 %35 to i64
  %37 = getelementptr inbounds i32, i32* %34, i64 %36
  %38 = load i32*, i32** %3, align 8
  %39 = load i32, i32* %6, align 4
  %40 = add nsw i32 %39, 1
  %41 = sext i32 %40 to i64
  %42 = getelementptr inbounds i32, i32* %38, i64 %41
  call void @swap(i32* %37, i32* %42)
  br label %43

; <label>:43:                                     ; preds = %33, %20
  br label %44

; <label>:44:                                     ; preds = %43
  %45 = load i32, i32* %6, align 4
  %46 = add nsw i32 %45, 1
  store i32 %46, i32* %6, align 4
  br label %13

; <label>:47:                                     ; preds = %13
  br label %48

; <label>:48:                                     ; preds = %47
  %49 = load i32, i32* %5, align 4
  %50 = add nsw i32 %49, 1
  store i32 %50, i32* %5, align 4
  br label %7
```

```
; <label>:51:                                     ; preds = %7
  ret void
}

; Function Attrs: noinline nounwind optnone uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  %3 = alloca i8*, align 8
  %4 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  %5 = call i32 (i8*, ...) @__isoc99_scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i32 0, i32 0),
i32* %2)
  %6 = load i32, i32* %2, align 4
  %7 = zext i32 %6 to i64
  %8 = call i8* @llvm.stacksave()
  store i8* %8, i8** %3, align 8
  %9 = alloca i32, i64 %7, align 16
  store i32 0, i32* %4, align 4
  br label %10

; <label>:10:                                     ; preds = %19, %0
  %11 = load i32, i32* %4, align 4
  %12 = load i32, i32* %2, align 4
  %13 = icmp slt i32 %11, %12
  br i1 %13, label %14, label %22

; <label>:14:                                     ; preds = %10
  %15 = load i32, i32* %4, align 4
  %16 = sext i32 %15 to i64
  %17 = getelementptr inbounds i32, i32* %9, i64 %16
  %18 = call i32 (i8*, ...) @__isoc99_scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i32 0, i32 0),
i32* %17)
  br label %19

; <label>:19:                                     ; preds = %14
  %20 = load i32, i32* %4, align 4
  %21 = add nsw i32 %20, 1
  store i32 %21, i32* %4, align 4
  br label %10

; <label>:22:                                     ; preds = %10
  %23 = load i32, i32* %2, align 4
  call void @sort(i32* %9, i32 %23)
  store i32 0, i32* %4, align 4
  br label %24

; <label>:24:                                     ; preds = %34, %22
  %25 = load i32, i32* %4, align 4
```

```
  %26 = load i32, i32* %2, align 4
  %27 = icmp slt i32 %25, %26
  br i1 %27, label %28, label %37

; <label>:28:                                     ; preds = %24
  %29 = load i32, i32* %4, align 4
  %30 = sext i32 %29 to i64
  %31 = getelementptr inbounds i32, i32* %9, i64 %30
  %32 = load i32, i32* %31, align 4
  %33 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @.str.1, i32 0, i32 0), i32 %32)
  br label %34

; <label>:34:                                     ; preds = %28
  %35 = load i32, i32* %4, align 4
  %36 = add nsw i32 %35, 1
  store i32 %36, i32* %4, align 4
  br label %24

; <label>:37:                                     ; preds = %24
  %38 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([2 x i8], [2 x i8]* @.str.2, i32 0, i32 0))
  store i32 0, i32* %1, align 4
  %39 = load i8*, i8** %3, align 8
  call void @llvm.stackrestore(i8* %39)
  %40 = load i32, i32* %1, align 4
  ret i32 %40
}

declare i32 @__isoc99_scanf(i8*, ...) #1

; Function Attrs: nounwind
declare i8* @llvm.stacksave() #2

declare i32 @printf(i8*, ...) #1

; Function Attrs: nounwind
declare void @llvm.stackrestore(i8*) #2

attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-divide-sqrt-fp-math"="false"
"disable-tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-elim"="true"
"no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false"
"no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="false"
"stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+fxsr,+mmx,+sse,+sse2,+x87"
"unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable-tail-calls"="false"
"less-precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-non-leaf"
"no-infs-fp-math"="false" "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false"
"no-trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="x86-64"
"target-features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #2 = { nounwind }
```

```
!llvm.module.flags = !{!0}
!llvm.ident = !{!1}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{!"clang version 6.0.0-1ubuntu2 (tags/RELEASE_600/final)"}
```

3. Factorial of number:

```
; ModuleID = 'factorial.c'
source_filename = "factorial.c"
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-pc-linux-gnu"

@.str = private unnamed_addr constant [3 x i8] c"%u\00", align 1
@.str.1 = private unnamed_addr constant [23 x i8] c"Factorial of %u is %u\0A\00", align 1

; Function Attrs: noinline nounwind optnone uwtable
define i32 @factorial(i32) #0 {
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4
  store i32 %0, i32* %3, align 4
  %4 = load i32, i32* %3, align 4
  %5 = icmp eq i32 %4, 1
  br i1 %5, label %9, label %6

; <label>:6:                                ; preds = %1
  %7 = load i32, i32* %3, align 4
  %8 = icmp eq i32 %7, 0
  br i1 %8, label %9, label %10

; <label>:9:                                ; preds = %6, %1
  store i32 1, i32* %2, align 4
  br label %16

; <label>:10:                               ; preds = %6
  %11 = load i32, i32* %3, align 4
  %12 = load i32, i32* %3, align 4
  %13 = sub i32 %12, 1
  %14 = call i32 @factorial(i32 %13)
  %15 = mul i32 %11, %14
  store i32 %15, i32* %2, align 4
  br label %16

; <label>:16:                               ; preds = %10, %9
  %17 = load i32, i32* %2, align 4
  ret i32 %17
}

; Function Attrs: noinline nounwind optnone uwtable
define i32 @main() #0 {
```

```
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  %3 = call i32 (i8*, ...) @__isoc99_scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i32 0, i32 0),
i32* %2)
  %4 = load i32, i32* %2, align 4
  %5 = load i32, i32* %2, align 4
  %6 = call i32 @factorial(i32 %5)
  %7 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([23 x i8], [23 x i8]* @.str.1, i32 0, i32 0), i32 %4,
i32 %6)
  ret i32 0
}

declare i32 @__isoc99_scanf(i8*, ...) #1

declare i32 @printf(i8*, ...) #1

attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-divide-sqrt-fp-math"="false"
"disable-tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-elim"="true"
"no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false"
"no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="false"
"stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+fxsr,+mmx,+sse,+sse2,+x87"
"unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable-tail-calls"="false"
"less-precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-non-leaf"
"no-infs-fp-math"="false" "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false"
"no-trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="x86-64"
"target-features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }

!llvm.module.flags = !{!0}
!llvm.ident = !{!1}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{!"clang version 6.0.0-1ubuntu2 (tags/RELEASE_600/final)"}
```

4. Anagram detection:

```
; ModuleID = 'anagram.c'
source_filename = "anagram.c"
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-pc-linux-gnu"

@.str = private unnamed_addr constant [14 x i8] c"Strings %s %s\00", align 1
@.str.1 = private unnamed_addr constant [9 x i8] c"Anagram\0A\00", align 1
@.str.2 = private unnamed_addr constant [13 x i8] c"Not Anagram\0A\00", align 1

; Function Attrs: noinline nounwind optnone uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca [100 x i8], align 16
```

```
  %3 = alloca [100 x i8], align 16
  %4 = alloca [26 x i32], align 16
  %5 = alloca i32, align 4
  %6 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  %7 = getelementptr inbounds [100 x i8], [100 x i8]* %2, i32 0, i32 0
  %8 = getelementptr inbounds [100 x i8], [100 x i8]* %3, i32 0, i32 0
  %9 = call i32 (i8*, ...) @__isoc99_scanf(i8* getelementptr inbounds ([14 x i8], [14 x i8]* @.str, i32 0, i32 0),
i8* %7, i8* %8)
  %10 = bitcast [26 x i32]* %4 to i8*
  call void @llvm.memset.p0i8.i64(i8* %10, i8 0, i64 104, i32 16, i1 false)
  store i32 0, i32* %5, align 4
  br label %11

; <label>:11:                                     ; preds = %29, %0
  %12 = load i32, i32* %5, align 4
  %13 = sext i32 %12 to i64
  %14 = getelementptr inbounds [100 x i8], [100 x i8]* %2, i64 0, i64 %13
  %15 = load i8, i8* %14, align 1
  %16 = sext i8 %15 to i32
  %17 = icmp ne i32 %16, 0
  br i1 %17, label %18, label %32

; <label>:18:                                     ; preds = %11
  %19 = load i32, i32* %5, align 4
  %20 = sext i32 %19 to i64
  %21 = getelementptr inbounds [100 x i8], [100 x i8]* %2, i64 0, i64 %20
  %22 = load i8, i8* %21, align 1
  %23 = sext i8 %22 to i32
  %24 = sub nsw i32 %23, 97
  %25 = sext i32 %24 to i64
  %26 = getelementptr inbounds [26 x i32], [26 x i32]* %4, i64 0, i64 %25
  %27 = load i32, i32* %26, align 4
  %28 = add nsw i32 %27, 1
  store i32 %28, i32* %26, align 4
  br label %29

; <label>:29:                                     ; preds = %18
  %30 = load i32, i32* %5, align 4
  %31 = add nsw i32 %30, 1
  store i32 %31, i32* %5, align 4
  br label %11

; <label>:32:                                     ; preds = %11
  store i32 0, i32* %5, align 4
  br label %33

; <label>:33:                                     ; preds = %51, %32
  %34 = load i32, i32* %5, align 4
  %35 = sext i32 %34 to i64
```

```
  %36 = getelementptr inbounds [100 x i8], [100 x i8]* %3, i64 0, i64 %35
  %37 = load i8, i8* %36, align 1
  %38 = sext i8 %37 to i32
  %39 = icmp ne i32 %38, 0
  br i1 %39, label %40, label %54

; <label>:40:                                     ; preds = %33
  %41 = load i32, i32* %5, align 4
  %42 = sext i32 %41 to i64
  %43 = getelementptr inbounds [100 x i8], [100 x i8]* %3, i64 0, i64 %42
  %44 = load i8, i8* %43, align 1
  %45 = sext i8 %44 to i32
  %46 = sub nsw i32 %45, 97
  %47 = sext i32 %46 to i64
  %48 = getelementptr inbounds [26 x i32], [26 x i32]* %4, i64 0, i64 %47
  %49 = load i32, i32* %48, align 4
  %50 = add nsw i32 %49, -1
  store i32 %50, i32* %48, align 4
  br label %51

; <label>:51:                                     ; preds = %40
  %52 = load i32, i32* %5, align 4
  %53 = add nsw i32 %52, 1
  store i32 %53, i32* %5, align 4
  br label %33

; <label>:54:                                     ; preds = %33
  store i32 1, i32* %6, align 4
  store i32 0, i32* %5, align 4
  br label %55

; <label>:55:                                     ; preds = %66, %54
  %56 = load i32, i32* %5, align 4
  %57 = icmp slt i32 %56, 26
  br i1 %57, label %58, label %69

; <label>:58:                                     ; preds = %55
  %59 = load i32, i32* %5, align 4
  %60 = sext i32 %59 to i64
  %61 = getelementptr inbounds [26 x i32], [26 x i32]* %4, i64 0, i64 %60
  %62 = load i32, i32* %61, align 4
  %63 = icmp ne i32 %62, 0
  br i1 %63, label %64, label %65

; <label>:64:                                     ; preds = %58
  store i32 0, i32* %6, align 4
  br label %69

; <label>:65:                                     ; preds = %58
  br label %66
```

```
; <label>:66:                                     ; preds = %65
  %67 = load i32, i32* %5, align 4
  %68 = add nsw i32 %67, 1
  store i32 %68, i32* %5, align 4
  br label %55

; <label>:69:                                     ; preds = %64, %55
  %70 = load i32, i32* %6, align 4
  %71 = icmp eq i32 %70, 1
  br i1 %71, label %72, label %74

; <label>:72:                                     ; preds = %69
  %73 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([9 x i8], [9 x i8]* @.str.1, i32 0, i32 0))
  br label %76

; <label>:74:                                     ; preds = %69
  %75 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([13 x i8], [13 x i8]* @.str.2, i32 0, i32 0))
  br label %76

; <label>:76:                                     ; preds = %74, %72
  ret i32 0
}

declare i32 @__isoc99_scanf(i8*, ...) #1

; Function Attrs: argmemonly nounwind
declare void @llvm.memset.p0i8.i64(i8* nocapture writeonly, i8, i64, i32, i1) #2

declare i32 @printf(i8*, ...) #1

attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-divide-sqrt-fp-math"="false"
"disable-tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-elim"="true"
"no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false"
"no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="false"
"stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+fxsr,+mmx,+sse,+sse2,+x87"
"unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable-tail-calls"="false"
"less-precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-non-leaf"
"no-infs-fp-math"="false" "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false"
"no-trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="x86-64"
"target-features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #2 = { argmemonly nounwind }

!llvm.module.flags = !{!0}
!llvm.ident = !{!1}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{!"clang version 6.0.0-1ubuntu2 (tags/RELEASE_600/final)"}
```

5. Square root using binary search:

```
; ModuleID = 'sqrt.c'
source_filename = "sqrt.c"
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-pc-linux-gnu"

@.str = private unnamed_addr constant [3 x i8] c"%f\00", align 1
@.str.1 = private unnamed_addr constant [20 x i8] c"Square root is %f \0A\00", align 1

; Function Attrs: noinline nounwind optnone uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca float, align 4
  %3 = alloca float, align 4
  %4 = alloca float, align 4
  %5 = alloca float, align 4
  store i32 0, i32* %1, align 4
  %6 = call i32 (i8*, ...) @__isoc99_scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i32 0, i32 0),
float* %2)
  store float 0.000000e+00, float* %3, align 4
  %7 = load float, float* %2, align 4
  store float %7, float* %4, align 4
  br label %8

; <label>:8:                                ; preds = %30, %0
  %9 = load float, float* %4, align 4
  %10 = load float, float* %3, align 4
  %11 = fsub float %9, %10
  %12 = fpext float %11 to double
  %13 = fcmp ogt double %12, 1.000000e-04
  br i1 %13, label %14, label %31

; <label>:14:                               ; preds = %8
  %15 = load float, float* %3, align 4
  %16 = load float, float* %4, align 4
  %17 = load float, float* %3, align 4
  %18 = fsub float %16, %17
  %19 = fdiv float %18, 2.000000e+00
  %20 = fadd float %15, %19
  store float %20, float* %5, align 4
  %21 = load float, float* %5, align 4
  %22 = load float, float* %5, align 4
  %23 = fmul float %21, %22
  %24 = load float, float* %2, align 4
  %25 = fcmp ogt float %23, %24
  br i1 %25, label %26, label %28

; <label>:26:                               ; preds = %14
  %27 = load float, float* %5, align 4
```

```
  store float %27, float* %4, align 4
  br label %30

; <label>:28:                               ; preds = %14
  %29 = load float, float* %5, align 4
  store float %29, float* %3, align 4
  br label %30

; <label>:30:                               ; preds = %28, %26
  br label %8

; <label>:31:                               ; preds = %8
  %32 = load float, float* %3, align 4
  %33 = fpext float %32 to double
  %34 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([20 x i8], [20 x i8]* @.str.1, i32 0, i32 0), double
%33)
  ret i32 0
}

declare i32 @__isoc99_scanf(i8*, ...) #1

declare i32 @printf(i8*, ...) #1

attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-divide-sqrt-fp-math"="false"
"disable-tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-elim"="true"
"no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false"
"no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="false"
"stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+fxsr,+mmx,+sse,+sse2,+x87"
"unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable-tail-calls"="false"
"less-precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-non-leaf"
"no-infs-fp-math"="false" "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false"
"no-trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="x86-64"
"target-features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }

!llvm.module.flags = !{!0}
!llvm.ident = !{!1}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{!"clang version 6.0.0-1ubuntu2 (tags/RELEASE_600/final)"}
```