DEGGENDORF INSTITUTE OF TECHNOLOGY

Master of Applied Computer Science

Formal languages and Compiler Construction - II (Practical)

Programming Lab SS20

Description of Improvement in “mytoyc” Compiler

Abhi Akbari - Student Number: 00802927

Vatsal Patel – Student Number: 00802146

10.07.2020

Table of Contents

1. Description of improvement 2

2. Outcome 5

3. Version specific changes 6

4. Contribution of Team member 7

LIST OF FIGURES

Figure No	Figure Description	Page No
`1	Original code of make file	6
2	Users can changes in code like this (Makefile)	6
3	Original code of codegen.hpp file	6
4	Users can changes in code like this (Codegen.hpp)	6

1. Description of improvement

In provided mytoyc compiler only integer type is defined and supported. With the reference of integer and other references, void type and double type added in given mytoyc compiler. Addition and modification described in the following report.

1 . tokens.l

In token file we introduce tokens and their symbolic names for void and double data type that is the first requirement to introduce a new type in compiler design. “VOID” and “DOUBLE” added as symbols which are considered by flex using symbols “void” and “double” from text or program code. These symbols (VOID, DOUBLE) being used as a terminal symbol in Grammar of Parser. Flex create external data object yytext (of String type) and store the value (by converting text string to value) of double (number with fractional point) read by flex from the text/program code. This string being converted to value and stored in yylval (object in yacc).

So here in this file we define lexeme, appropriate return attribute and relevant required assignment operation.

2. Parser.y

After making change in token file we need to reflect that in terminal list and grammar, so added terminals, non-terminals and enhanced production rules in this file.

Declaration:

So for this, in declaration part we define terminal symbols which are same token added in token.l file.

For support of double type value terminal, double type value (as a terminal) is required which is defined and type details added in union. double type variable declared inside YYSTYPE union. To return value of type double via yylval.

Provided mytoyc compiler just process integer data type. So to add other data type support as well it is necessity to know the type of data in addition to value of data.

In declaration part, non-terminal “type” declared (missing in t-0 branch) which used in Grammar/ Production rule to pass the type of data.

Production Rule:

For all Production rule, new node is created using new variadic node method (defined in node.hpp). This new node is created to achieve a goal of passing type information as well in addition to current implementation.

Rule for non-terminal “type” is added with possibilities to support data type void and double in addition to predefined integer type. In node creation fix string “void”, “int” and “double” passed to compare. So now on these two types are also allowed.

For non-terminal “numeric”, production rule extended to include double type. Numeric value assigned using NDouble function (defined in codegen.cpp).

3. codegen.hpp

In this file, which type of LLVM value return for every type is defined (here in this file int32 is already define for integer type).

So, we defined new function for void and double type and functionality of this function is to manage each type context and describe the return type for void and double type. In similar way to provided Integer type with reference of www.llvm.org.

4. node.hpp

Added variadic macro in order to create new node with variable multiple numbers of arguments.

For number, class is defined to fetch data value and represent or print the same on display in some format. This class for integer is provided. In similar way we revised it for double type.

Void type doesn't have any return value. So, for void type, constructor created in NReturnStatement Class to assign Null value to expression.

NFunctionDeclaration and NExternDeclaration class defined for integer type only, so previously type is not required as function consider numeric value to be integer and process accordingly, but after having double, type entity require to determine actual data type of number added for identifier.

So here in both class we define one extra variable as per parser. Ultimately, we defined one extra variable then we add one extra argument in method of these two class. So, whenever user write any function or any external function so every time it checks the type of data (whether it is internal or external).

5. codegen.cpp

Inside this file we define functionality of type function which we use for type checking. here in this function it every time compare the value from user's function to predefined string ("void", "int", "double") and from that comparison it decides which type of the function or variable object is. Here in this class we add value inside the condition of type function for type checking. So for that we add two if condition for void and double type.

NDouble function defined with reference of NInteger.

NReturnStatement have code statement for integer data only. So, using if-else loop return NULL statement included for void return type. It returns value for integer and double.

After change in NFunctionDeclaration and NExternDeclaration class (node.hpp), necessarily relevant change needed in function call for same. So when we take data type information from llvm in “FunctionType”, we changed it to generic from fix “getIntType()”. Same function defined for void (getVoidType) and double (getDoubleType) type in codegen.hpp.

2. Outcome

We added void and double type. We run the modified compiler on Lab Computer, it run successfully.

We didn’t used any special method and not added any special command.

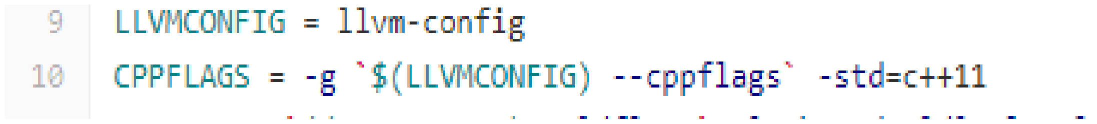
Compiler able to interpret added void and double type and provide appropriate output.

3. Version specific changes

Sometimes, It might be possible that user have some different version of libraries in their operating system and might be some functionality of different OS raise some compile time error while user try to run this project.

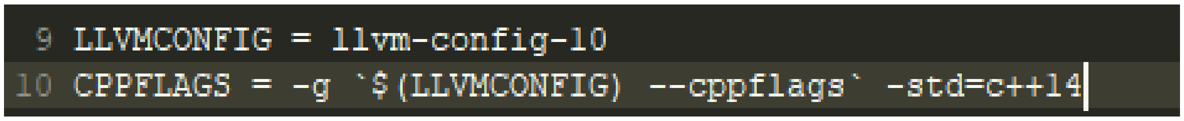
So, for that we looked for issue and figured out that below changes can solve the library issue or compilation version difference issue. We tried and it worked for us under different environment.

1.Makefile



```
LLVMCONFIG = llvm-config
CPPFLAGS = -g `$(LLVMCONFIG) --cppflags` -std=c++11
```

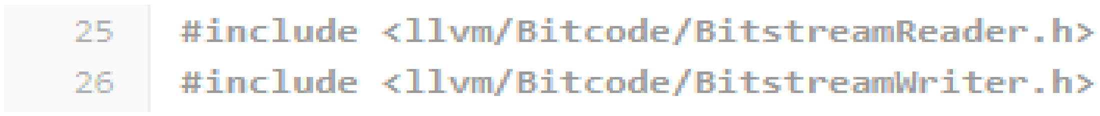
1. Original code of make file



```
LLVMCONFIG = llvm-config-10
CPPFLAGS = -g `$(LLVMCONFIG) --cppflags` -std=c++14
```

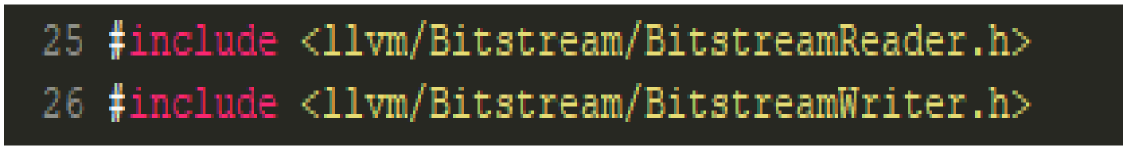
2. Users can change in code like this (Makefile)

2. Codegen.hpp



```
#include <llvm/Bitcode/BitstreamReader.h>
#include <llvm/Bitcode/BitstreamWriter.h>
```

3. Original code of codegen.hpp file



```
#include <llvm/Bitstream/BitstreamReader.h>
#include <llvm/Bitstream/BitstreamWriter.h>
```

4. Users can changes in code like this (Codegen.hpp)

4. Contribution of team member

Abhi Akbari: Introduced void type, Report writing, Version specific changes.

Vatsal Patel: Introduced Double type, Report writing.

Formally we started to look for 1-1 type (void – Abhi, double - Vatsal) to differentiate individual’s task but as project work goes on and some difficulty encountered, we worked together in later stage for all remaining stuff.