# IFCC

Generated by Doxygen 1.9.8

# Chapter 1

# Welcome to the Documentation

## 1.1 Overview

This project implements a C compiler with a focus on generating Intermediate Representation (IR), performing code analysis, and generating assembly code. It includes functionalities such as syntax checking, control flow graph (CFG) generation, and code optimization.

## 1.2 Features

- **C Syntax Analysis:** Parses C source code and performs syntax checks.
- **Intermediate Representation (IR):** Generates and manipulates IR for code analysis.
- **Assembly Generation:** Converts IR into assembly code for various platforms.
- **CFG Generation:** Generates control flow graphs for visualizing program execution.
- **Code Checking:** Validates the code for errors and potential optimizations.

## 1.3 How to Use

To compile and run the compiler:

1. Clone or download the repository.
2. Set up the build environment.
3. Compile the source code using `make` or the appropriate build command.
4. Run the compiler with the C source file as an argument:
   `./ifcc path/to/file.c`

## 1.4 Dependencies

- **ANTLR:** For parsing C source code.
- **Graphviz:** For generating CFG visualizations.

## 1.5 Project Structure

- **src/:** Source code for the compiler.
- **include/:** Header files defining the compiler's functionality.
- **test/:** Test files for validating the compiler's correctness.
- **docs/:** Documentation for the project.
- **README.md:** Project overview and setup instructions.

## 1.6 License

This project is licensed under the MIT License.

# Chapter 2

# Hierarchical Index

## 2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

# Chapter 3

# Class Index

## 3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 4

# File Index

## 4.1 File List

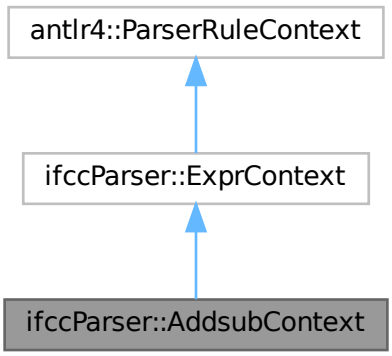
Here is a list of all documented files with brief descriptions:

# Chapter 5

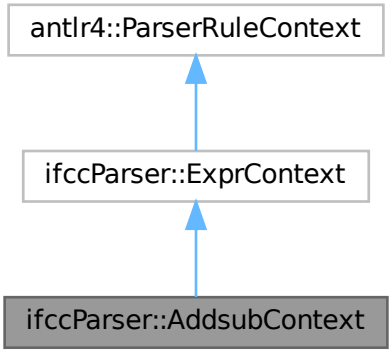
# Class Documentation

## 5.1 ifccParser::AddsubContext Class Reference

Inheritance diagram for ifccParser::AddsubContext:



Collaboration diagram for ifccParser::AddsubContext:




#### Public Member Functions

- **AddsubContext** (ExprContext ∗ctx)
- std::vector< ExprContext ∗ > **expr** ()
- ExprContext ∗ **expr** (size_t i)
- virtual std::any **accept** (antlr4::tree::ParseTreeVisitor ∗visitor) override

### Public Member Functions inherited from ifccParser::ExprContext

- **ExprContext** (antlr4::ParserRuleContext ∗parent, size_t invokingState)
- void **copyFrom** (ExprContext ∗context)
- virtual size_t **getRuleIndex** () const override

#### Public Attributes

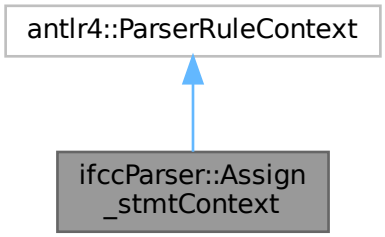
- antlr4::Token ∗ **OP** = nullptr

The documentation for this class was generated from the following files:

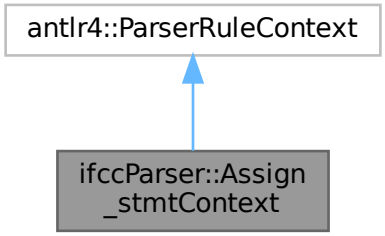
- ifccParser.h
- ifccParser.cpp

## 5.2 ifccParser::Assign_stmtContext Class Reference

Inheritance diagram for ifccParser::Assign_stmtContext:




Collaboration diagram for ifccParser::Assign_stmtContext:




### Public Member Functions

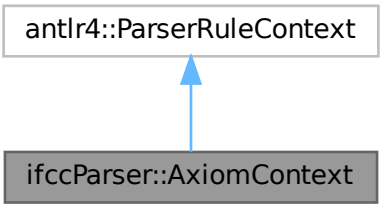
- **Assign_stmtContext** (antlr4::ParserRuleContext ∗parent, size_t invokingState)
- virtual size_t **getRuleIndex** () const override
- antlr4::tree::TerminalNode ∗ **VAR** ()
- ExprContext ∗ **expr** ()
- virtual std::any **accept** (antlr4::tree::ParseTreeVisitor ∗visitor) override

The documentation for this class was generated from the following files:

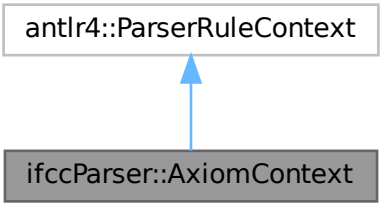
- ifccParser.h
- ifccParser.cpp

## 5.3 ifccParser::AxiomContext Class Reference

Inheritance diagram for ifccParser::AxiomContext:




Collaboration diagram for ifccParser::AxiomContext:




### Public Member Functions

- **AxiomContext** (antlr4::ParserRuleContext ∗parent, size_t invokingState)
- virtual size_t **getRuleIndex** () const override
- ProgContext ∗ **prog** ()
- antlr4::tree::TerminalNode ∗ **EOF** ()
- virtual std::any **accept** (antlr4::tree::ParseTreeVisitor ∗visitor) override

The documentation for this class was generated from the following files:

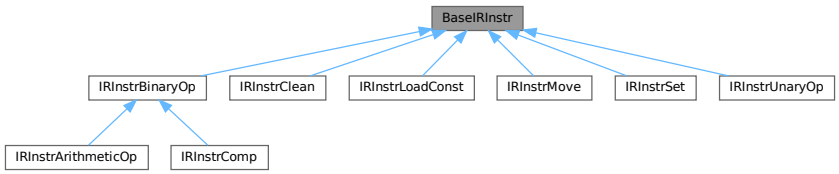
- ifccParser.h
- ifccParser.cpp

## 5.4 BaseIRInstr Class Reference

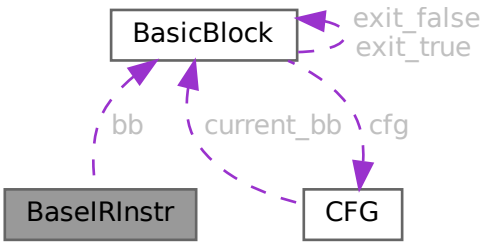
Represents a base class for intermediate representation instructions.

`#include <BaseIRInstr.h>`

Inheritance diagram for BaseIRInstr:




Collaboration diagram for BaseIRInstr:




### Public Member Functions

- BaseIRInstr (BasicBlock *bb_)

  *Constructs an instruction for a given basic block.*
- BasicBlock * getBB ()

  *Gets the basic block that this instruction belongs to.*
- virtual void gen_asm (ostream &o)=0

  *Generates the assembly code for this instruction.*

### Protected Attributes

- BasicBlock * bb

  *The basic block that this instruction belongs to.*

### 5.4.1 Detailed Description

Represents a base class for intermediate representation instructions.

This class serves as the base for all instruction types in the intermediate representation (IR). It provides a basic structure for handling assembly generation and access to the basic block that the instruction belongs to.

### 5.4.2 Constructor & Destructor Documentation

#### 5.4.2.1 BaseIRInstr()

```
BaseIRInstr::BaseIRInstr (
            BasicBlock * bb_ )  [inline]
```

Constructs an instruction for a given basic block.

Initializes the instruction with the basic block it belongs to.

**Parameters**

| | |
|---|---|
| *bb_* | The basic block to which this instruction belongs. |

### 5.4.3 Member Function Documentation

#### 5.4.3.1 gen_asm()

```
virtual void BaseIRInstr::gen_asm (
            ostream & o )  [pure virtual]
```

Generates the assembly code for this instruction.

This is a pure virtual function that must be implemented by derived classes to generate the specific assembly code for each type of instruction.

**Parameters**

| | |
|---|---|
| *o* | The output stream where the generated assembly code will be written. |

Implemented in IRInstrArithmeticOp, IRInstrUnaryOp, IRInstrComp, IRInstrLoadConst, IRInstrMove, and IRInstrBinaryOp.

#### 5.4.3.2 getBB()

```
BasicBlock * BaseIRInstr::getBB ( )
```

Gets the basic block that this instruction belongs to.

**Returns**

The basic block associated with this instruction.

### 5.4.4 Member Data Documentation

#### 5.4.4.1 bb

```
BasicBlock* BaseIRInstr::bb  [protected]
```

The basic block that this instruction belongs to.

The documentation for this class was generated from the following files:

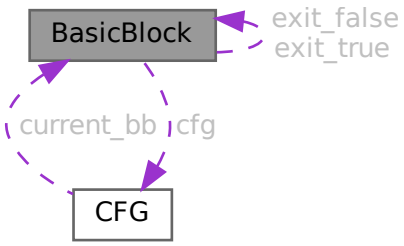
- BaseIRInstr.h
- BaseIRInstr.cpp

## 5.5 BasicBlock Class Reference

Represents a basic block in the control flow graph (CFG).

```
#include <BasicBlock.h>
```

Collaboration diagram for BasicBlock:




**Public Member Functions**

- BasicBlock (CFG *cfg, string entry_label)

  *Constructs a BasicBlock with the given CFG and entry label.*
- void gen_asm (ostream &o)

  *Generates the assembly code for this basic block.*
- void add_IRInstr (BaseIRInstr *instr)

  *Adds an instruction to the basic block.*
- CFG * getCFG ()

  *Gets the CFG associated with this basic block.*

- string getLabel ()

  *Retrieves the label associated with the current block.*
- vector< BaseIRInstr * > getInstr ()

  *Retrieves the list of instructions within the current block.*
- void setExitTrue (BasicBlock *bb)

  *Sets the "true" exit point for the current block.*
- void setExitFalse (BasicBlock *bb)

  *Sets the "false" exit point for the current block.*
- BasicBlock * getExitTrue ()

  *Retrieves the "true" exit point of the current block.*
- BasicBlock * getExitFalse ()

  *Retrieves the "false" exit point of the current block.*

**Protected Attributes**

- BasicBlock * exit_true
- BasicBlock * exit_false
- string label

  *The label for the basic block, also used as the label in the generated assembly code.*
- CFG * cfg

  *The control flow graph to which this basic block belongs.*
- vector< BaseIRInstr * > instrs

  *A vector of instructions that belong to this basic block.*

### 5.5.1 Detailed Description

Represents a basic block in the control flow graph (CFG).

A basic block is a sequence of instructions with a single entry point and a single exit point. This class is responsible for managing the instructions in the block and generating the assembly code.

### 5.5.2 Constructor & Destructor Documentation

#### 5.5.2.1 BasicBlock()

```
BasicBlock::BasicBlock (
            CFG * cfg,
            string entry_label )
```

Constructs a BasicBlock with the given CFG and entry label.

Initializes a new basic block with a label and associates it with a specific control flow graph (CFG).

**Parameters**

| | |
|---|---|
| *cfg* | The CFG where this basic block belongs. |
| *entry_label* | The entry label for the basic block. |

### 5.5.3 Member Function Documentation

#### 5.5.3.1 add_IRInstr()

```
void BasicBlock::add_IRInstr (
            BaseIRInstr * instr )
```

Adds an instruction to the basic block.

This method adds a new intermediate representation instruction (IRInstr) to the basic block.

**Parameters**

| | |
|---|---|
| *instr* | A pointer to the instruction to add. |

#### 5.5.3.2 gen_asm()

```
void BasicBlock::gen_asm (
            ostream & o )
```

Generates the assembly code for this basic block.

This method generates the x86 assembly code for all the instructions in the basic block.

**Parameters**

| | |
|---|---|
| *o* | The output stream where the assembly code will be written. |

#### 5.5.3.3 getCFG()

```
CFG * BasicBlock::getCFG ( )
```

Gets the CFG associated with this basic block.

This method retrieves the CFG to which this basic block belongs.

**Returns**

A pointer to the CFG associated with this basic block.

#### 5.5.3.4 getExitFalse()

```
BasicBlock * BasicBlock::getExitFalse ( )
```

Retrieves the "false" exit point of the current block.

This function returns the basic block representing the "false" exit point in the control flow. It is used to identify where the program flow should continue if a condition evaluates to false.

**Returns**

A pointer to the BasicBlock representing the "false" exit.

#### 5.5.3.5 getExitTrue()

```
BasicBlock * BasicBlock::getExitTrue ( )
```

Retrieves the "true" exit point of the current block.

This function returns the basic block representing the "true" exit point in the control flow. It is used to identify where the program flow should continue if a condition evaluates to true.

**Returns**

A pointer to the `BasicBlock` representing the "true" exit.

#### 5.5.3.6 getInstr()

```
vector< BaseIRInstr * > BasicBlock::getInstr ( )
```

Retrieves the list of instructions within the current block.

This function returns a vector containing all the instructions present in this block. Each instruction represents a basic operation in the program's flow, used in program analysis or code generation.

**Returns**

A vector of `BaseIRInstr *` representing the instructions in the block.

#### 5.5.3.7 getLabel()

```
string BasicBlock::getLabel ( )
```

Retrieves the label associated with the current block.

This function returns the label associated with this particular control flow block. The label is typically used to uniquely identify this block in control flow analysis.

**Returns**

A string representing the label of this control flow block.

#### 5.5.3.8 setExitFalse()

```
void BasicBlock::setExitFalse (
            BasicBlock * bb )
```

Sets the "false" exit point for the current block.

This function sets the block that serves as the "false" exit in the control flow of the program. This is typically used for conditional branches, where the program flow follows one path if a condition is true, and another path if it is false.

**Parameters**

| | |
|---|---|
| *bb* | A pointer to the BasicBlock that represents the "false" exit of the current block. |

#### 5.5.3.9 setExitTrue()

```
void BasicBlock::setExitTrue (
            BasicBlock * bb )
```

Sets the "true" exit point for the current block.

This function sets the block that serves as the "true" exit in the control flow of the program. This is often used when there is a conditional branch and the flow of execution diverges depending on the outcome of a condition.

**Parameters**

| | |
|---|---|
| *bb* | A pointer to the BasicBlock that represents the "true" exit of the current block. |

### 5.5.4 Member Data Documentation

#### 5.5.4.1 cfg

```
CFG* BasicBlock::cfg  [protected]
```

The control flow graph to which this basic block belongs.

#### 5.5.4.2 exit_false

```
BasicBlock* BasicBlock::exit_false  [protected]
```

Pointer to the basic block representing the "false" exit. This is used for the branch or conditional statement when the condition is false. It can be null if no "false" exit exists.

#### 5.5.4.3 exit_true

```
BasicBlock* BasicBlock::exit_true  [protected]
```

Pointer to the basic block representing the "true" exit. This is used for the branch or conditional statement when the condition is true. It can be null if no "true" exit exists.

#### 5.5.4.4 instrs

```
vector<BaseIRInstr *> BasicBlock::instrs  [protected]
```

A vector of instructions that belong to this basic block.

#### 5.5.4.5 label

```
string BasicBlock::label  [protected]
```

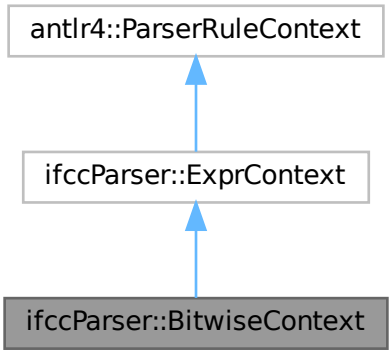
The label for the basic block, also used as the label in the generated assembly code.

The documentation for this class was generated from the following files:

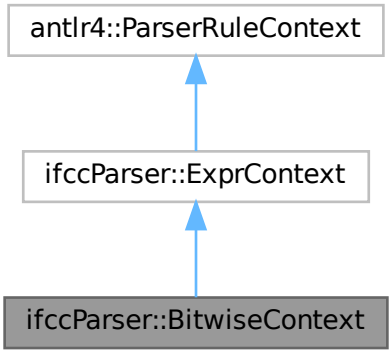
- BasicBlock.h
- BasicBlock.cpp

## 5.6 ifccParser::BitwiseContext Class Reference

Inheritance diagram for ifccParser::BitwiseContext:




Collaboration diagram for ifccParser::BitwiseContext:



#### Public Member Functions

- **BitwiseContext** (ExprContext ∗ctx)
- std::vector< ExprContext ∗ > **expr** ()
- ExprContext ∗ **expr** (size_t i)
- virtual std::any **accept** (antlr4::tree::ParseTreeVisitor ∗visitor) override

### Public Member Functions inherited from ifccParser::ExprContext

- **ExprContext** (antlr4::ParserRuleContext ∗parent, size_t invokingState)
- void **copyFrom** (ExprContext ∗context)
- virtual size_t **getRuleIndex** () const override

#### Public Attributes

- antlr4::Token ∗ **OP** = nullptr

The documentation for this class was generated from the following files:

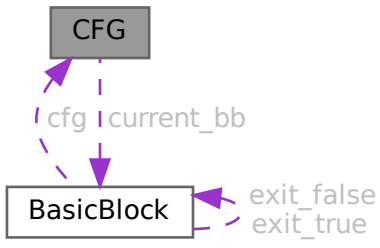
- ifccParser.h
- ifccParser.cpp

## 5.7 CFG Class Reference

Represents a Control Flow Graph (CFG) in an Intermediate Representation (IR).

`#include <CFG.h>`

Collaboration diagram for CFG:



### Public Member Functions

- CFG (string label, map< string, int > SymbolIndex, int initialNextFreeSymbolIndex)

  *Constructs a CFG with the given label and symbol index information.*
- void add_bb (BasicBlock *bb)

  *Adds a basic block to the control flow graph.*
- void gen_asm (ostream &o)

  *Generates the assembly code for the entire control flow graph.*
- void gen_asm_prologue (ostream &o)

  *Generates the assembly prologue for the control flow graph.*
- void gen_asm_epilogue (ostream &o)

  *Generates the assembly epilogue for the control flow graph.*
- string create_new_tempvar ()

  *Creates a new temporary variable in the control flow graph.*
- int get_var_index (string name)

  *Retrieves the index of a variable by its name.*
- BasicBlock * getCurrentBasicBlock ()

  *Retrieves the current basic block in the control flow graph.*
- void setCurrentBasicBlock (BasicBlock *bb)

  *Sets the current basic block in the control flow graph.*
- void resetNextFreeSymbolIndex ()

  *Resets the next free symbol index to its initial value.*
- void gen_cfg_graphviz (ostream &o)

  *Generates the Graphviz representation of the control flow graph (CFG).*
- string getLabel ()

  *Retrieves the label associated with the control flow graph (CFG).*

### Protected Attributes

- map< string, int > SymbolIndex

  *A map of symbol names to their respective indices.*
- int nextFreeSymbolIndex

  *The next available symbol index.*
- const int initialTempPos

  *The initial value for the next free symbol index.*
- vector< BasicBlock * > bbs

  *A vector containing all the basic blocks in the control flow graph.*
- BasicBlock * current_bb

  *A pointer to the current basic block being processed.*
- string label

  *The label associated with the control flow graph.*

## 5.7.1 Detailed Description

Represents a Control Flow Graph (CFG) in an Intermediate Representation (IR).

A CFG consists of basic blocks and represents the flow of control in a program. This class is responsible for managing the basic blocks and generating assembly code corresponding to the control flow graph.

## 5.7.2 Constructor & Destructor Documentation

### 5.7.2.1 CFG()

```
CFG::CFG (
            string label,
            map< string, int > SymbolIndex,
            int initialNextFreeSymbolIndex )
```

Constructs a CFG with the given label and symbol index information.

Initializes the control flow graph with a label and symbol index, and sets the initial index for the next free symbol.

**Parameters**

| | |
|---|---|
| *label* | The label for the CFG. |
| *SymbolIndex* | A map that maps symbol names to their respective indices. |
| *initialNextFreeSymbolIndex* | The initial value for the next free symbol index. |

## 5.7.3 Member Function Documentation

### 5.7.3.1 add_bb()

```
void CFG::add_bb (
            BasicBlock * bb )
```

Adds a basic block to the control flow graph.

This method adds a new basic block to the vector of basic blocks that make up the CFG.

**Parameters**

| | |
|---|---|
| *bb* | A pointer to the basic block to add. |

### 5.7.3.2 create_new_tempvar()

```
string CFG::create_new_tempvar ( )
```

Creates a new temporary variable in the control flow graph.

This method creates a new temporary variable with a unique name and returns the name of the variable.

**Returns**

The name of the new temporary variable.

#### 5.7.3.3 gen_asm()

```
void CFG::gen_asm (
            ostream & o )
```

Generates the assembly code for the entire control flow graph.

This method generates the assembly code for all basic blocks in the CFG.

**Parameters**

| | |
|---|---|
| *o* | The output stream where the assembly code will be written. |

#### 5.7.3.4 gen_asm_epilogue()

```
void CFG::gen_asm_epilogue (
            ostream & o )
```

Generates the assembly epilogue for the control flow graph.

This method generates the final assembly code that cleans up the environment after the program has executed.

**Parameters**

| | |
|---|---|
| *o* | The output stream where the epilogue will be written. |

#### 5.7.3.5 gen_asm_prologue()

```
void CFG::gen_asm_prologue (
            ostream & o )
```

Generates the assembly prologue for the control flow graph.

This method generates the initial assembly code that sets up the environment for the program to start execution.

**Parameters**

| | |
|---|---|
| *o* | The output stream where the prologue will be written. |

#### 5.7.3.6 gen_cfg_graphviz()

```
void CFG::gen_cfg_graphviz (
            ostream & o )
```

Generates the Graphviz representation of the control flow graph (CFG).

This function generates a Graphviz-compatible `.dot` file that visualizes the control flow of the program. The `.dot` file describes the nodes (representing basic blocks or instructions) and edges (representing the flow of control between the blocks) of the control flow graph.

The generated Graphviz representation is written to the provided output stream.

**Parameters**

| | |
|---|---|
| *o* | The output stream to which the Graphviz `.dot` representation of the CFG is written. This is typically a file stream (e.g., `ofstream`) that writes to a `.dot` file. |

#### 5.7.3.7 get_var_index()

```
int CFG::get_var_index (
            string name )
```

Retrieves the index of a variable by its name.

This method retrieves the index of the variable in the symbol table.

**Parameters**

| | |
|---|---|
| *name* | The name of the variable. |

**Returns**

The index of the variable, or -1 if the variable does not exist.

#### 5.7.3.8 getCurrentBasicBlock()

```
BasicBlock * CFG::getCurrentBasicBlock ( )
```

Retrieves the current basic block in the control flow graph.

This method returns a pointer to the current basic block being processed.

**Returns**

A pointer to the current basic block.

#### 5.7.3.9 getLabel()

```
string CFG::getLabel ( )
```

Retrieves the label associated with the control flow graph (CFG).

This function returns a string label that represents the name or identifier associated with the current control flow graph. The label can be used to identify different parts of the program, such as functions or basic blocks.

The label is typically used for naming the nodes and edges in the Graphviz `.dot` representation or for other program analysis purposes.

**Returns**

A string representing the label of the control flow graph. This could be a function name, block identifier, or any other relevant label.

#### 5.7.3.10 resetNextFreeSymbolIndex()

```
void CFG::resetNextFreeSymbolIndex ( )
```

Resets the next free symbol index to its initial value.

This method resets the index for the next free symbol to its initial state.

#### 5.7.3.11 setCurrentBasicBlock()

```
void CFG::setCurrentBasicBlock (
            BasicBlock * bb )
```

Sets the current basic block in the control flow graph.

This method sets the basic block that is currently being processed in the CFG.

**Parameters**

| | |
|---|---|
| *bb* | A pointer to the basic block to set as the current block. |

### 5.7.4 Member Data Documentation

#### 5.7.4.1 bbs

```
vector<BasicBlock *> CFG::bbs  [protected]
```

A vector containing all the basic blocks in the control flow graph.

#### 5.7.4.2 current_bb

```
BasicBlock* CFG::current_bb  [protected]
```

A pointer to the current basic block being processed.

#### 5.7.4.3 initialTempPos

```
const int CFG::initialTempPos  [protected]
```

The initial value for the next free symbol index.

#### 5.7.4.4 label

```
string CFG::label  [protected]
```

The label associated with the control flow graph.

#### 5.7.4.5 nextFreeSymbolIndex

```
int CFG::nextFreeSymbolIndex  [protected]
```

The next available symbol index.

#### 5.7.4.6 SymbolIndex

```
map<string, int> CFG::SymbolIndex  [protected]
```

A map of symbol names to their respective indices.

The documentation for this class was generated from the following files:

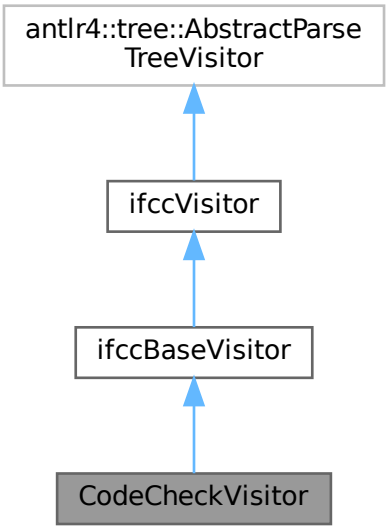
- CFG.h
- CFG.cpp

## 5.8 CodeCheckVisitor Class Reference

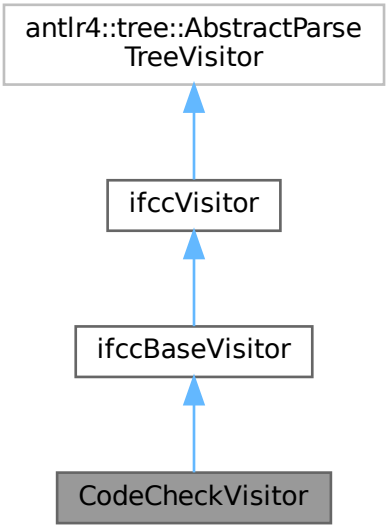
A visitor class for checking code correctness.

```
#include <CodeCheckVisitor.h>
```

Inheritance diagram for CodeCheckVisitor:



Collaboration diagram for CodeCheckVisitor:




### Public Member Functions

- virtual antlrcpp::Any visitReturn_stmt (ifccParser::Return_stmtContext ∗ctx) override

  *Visits a return statement in the parsed code.*
- virtual antlrcpp::Any visitAssign_stmt (ifccParser::Assign_stmtContext ∗ctx) override

  *Visits an assignment statement in the parsed code.*
- virtual antlrcpp::Any visitDecl_stmt (ifccParser::Decl_stmtContext ∗ctx) override

  *Visits a declaration statement in the parsed code.*
- antlrcpp::Any visitExpr (ifccParser::ExprContext ∗expr)

  *Visits any expression in the parsed code.*
- virtual antlrcpp::Any visitAddsub (ifccParser::AddsubContext ∗ctx) override

  *Visits an addition or subtraction expression.*
- virtual antlrcpp::Any visitMuldiv (ifccParser::MuldivContext ∗ctx) override

  *Visits a multiplication or division expression.*
- virtual antlrcpp::Any visitBitwise (ifccParser::BitwiseContext ∗ctx) override

  *Visits a bitwise operation expression.*
- virtual antlrcpp::Any visitComp (ifccParser::CompContext ∗ctx) override

  *Visits a comparison expression.*
- virtual antlrcpp::Any visitUnary (ifccParser::UnaryContext ∗ctx) override

  *Visits a unary expression.*
- virtual antlrcpp::Any visitPre (ifccParser::PreContext ∗ctx) override

  *Visits a pre-unary operation (e.g., prefix increment/decrement).*
- virtual antlrcpp::Any visitPost (ifccParser::PostContext ∗ctx) override

  *Visits a post-unary operation (e.g., postfix increment/decrement).*
- map< string, int > getSymbolsTable () const

  *Retrieves the symbols table, which holds variable names and their associated offsets.*

- map< string, bool > getIsUsed () const

  *Retrieves the map indicating whether variables are used in the code.*
- int getCurrentOffset () const

  *Retrieves the current offset used in the variable symbol table.*

## Public Member Functions inherited from ifccBaseVisitor

- virtual std::any visitAxiom (ifccParser::AxiomContext *ctx) override
- virtual std::any visitProg (ifccParser::ProgContext *ctx) override
- virtual std::any visitStatement (ifccParser::StatementContext *ctx) override
- virtual std::any visitIncrdecr_stmt (ifccParser::Incrdecr_stmtContext *ctx) override
- virtual std::any visitPar (ifccParser::ParContext *ctx) override
- virtual std::any visitConst (ifccParser::ConstContext *ctx) override
- virtual std::any visitVar (ifccParser::VarContext *ctx) override

### Protected Attributes

- map< string, int > symbolsTable

  *The symbols table containing variable names and their offsets.*
- map< string, bool > isUsed

  *A map indicating if a variable has been used in the code.*
- map< string, bool > hasAValue

  *A map to track whether a variable has been assigned a value.*
- int currentOffset = 0

  *The current offset value used for the symbols table.*

## 5.8.1 Detailed Description

A visitor class for checking code correctness.

This class extends the `ifccBaseVisitor` and is responsible for checking various aspects of code correctness, including variable declarations, assignments, and expressions, during the parsing phase of the compiler.

## 5.8.2 Member Function Documentation

### 5.8.2.1 getCurrentOffset()

```
int CodeCheckVisitor::getCurrentOffset ( ) const  [inline]
```

Retrieves the current offset used in the variable symbol table.

#### Returns

The current offset value.

#### 5.8.2.2 getIsUsed()

```
map< string, bool > CodeCheckVisitor::getIsUsed ( ) const  [inline]
```

Retrieves the map indicating whether variables are used in the code.

**Returns**

The map of variables and their usage status.

#### 5.8.2.3 getSymbolsTable()

```
map< string, int > CodeCheckVisitor::getSymbolsTable ( ) const  [inline]
```

Retrieves the symbols table, which holds variable names and their associated offsets.

**Returns**

The symbols table as a map of variable names and their associated offsets.

#### 5.8.2.4 visitAddsub()

```
antlrcpp::Any CodeCheckVisitor::visitAddsub (
            ifccParser::AddsubContext * ctx )  [override], [virtual]
```

Visits an addition or subtraction expression.

This method processes expressions involving addition or subtraction and checks for correctness in terms of variable usage.

**Parameters**

| | |
|---|---|
| *ctx* | The context for the addition or subtraction expression. |

**Returns**

A result of the visit, typically unused.

Reimplemented from ifccBaseVisitor.

#### 5.8.2.5 visitAssign_stmt()

```
antlrcpp::Any CodeCheckVisitor::visitAssign_stmt (
            ifccParser::Assign_stmtContext * ctx )  [override], [virtual]
```

Visits an assignment statement in the parsed code.

This method checks whether variables are assigned values correctly and whether the variables involved are defined.

**Parameters**

| | |
|---|---|
| *ctx* | The context for the assignment statement. |

**Returns**

A result of the visit, typically unused.

Reimplemented from ifccBaseVisitor.

#### 5.8.2.6 visitBitwise()

```
antlrcpp::Any CodeCheckVisitor::visitBitwise (
            ifccParser::BitwiseContext * ctx )  [override], [virtual]
```

Visits a bitwise operation expression.

This method processes bitwise operations like AND, OR, XOR, etc., and ensures that all variables in these operations are declared.

**Parameters**

| | |
|---|---|
| *ctx* | The context for the bitwise operation expression. |

**Returns**

A result of the visit, typically unused.

Reimplemented from ifccBaseVisitor.

#### 5.8.2.7 visitComp()

```
antlrcpp::Any CodeCheckVisitor::visitComp (
            ifccParser::CompContext * ctx )  [override], [virtual]
```

Visits a comparison expression.

This method processes comparison operations like equality, inequality, greater than, less than, etc., and checks for any correctness issues regarding the variables used.

**Parameters**

| | |
|---|---|
| *ctx* | The context for the comparison expression. |

**Returns**

A result of the visit, typically unused.

Reimplemented from ifccBaseVisitor.

#### 5.8.2.8 visitDecl_stmt()

```
antlrcpp::Any CodeCheckVisitor::visitDecl_stmt (
            ifccParser::Decl_stmtContext * ctx )  [override], [virtual]
```

Visits a declaration statement in the parsed code.

This method ensures that the variable being declared is correctly handled, ensuring no variable is declared multiple times, or is used before being declared.

**Parameters**

| | |
|---|---|
| *ctx* | The context for the declaration statement. |

**Returns**

A result of the visit, typically unused.

Reimplemented from ifccBaseVisitor.

#### 5.8.2.9 visitExpr()

```
antlrcpp::Any CodeCheckVisitor::visitExpr (
            ifccParser::ExprContext * expr )
```

Visits any expression in the parsed code.

This method processes expressions and checks whether variables in expressions are valid and properly declared.

**Parameters**

| | |
|---|---|
| *expr* | The expression context to check. |

**Returns**

A result of the visit, typically unused.

#### 5.8.2.10 visitMuldiv()

```
antlrcpp::Any CodeCheckVisitor::visitMuldiv (
            ifccParser::MuldivContext * ctx )  [override], [virtual]
```

Visits a multiplication or division expression.

This method processes multiplication and division expressions and checks whether variables involved are correctly declared.

**Parameters**

| | |
|---|---|
| *ctx* | The context for the multiplication or division expression. |

**Returns**

A result of the visit, typically unused.

Reimplemented from ifccBaseVisitor.

#### 5.8.2.11 visitPost()

```
antlrcpp::Any CodeCheckVisitor::visitPost (
            ifccParser::PostContext * ctx )  [override], [virtual]
```

Visits a post-unary operation (e.g., postfix increment/decrement).

This method processes post-unary operations and checks for correctness in usage.

**Parameters**

| | |
|---|---|
| *ctx* | The context for the post-unary expression. |

**Returns**

A result of the visit, typically unused.

Reimplemented from ifccBaseVisitor.

#### 5.8.2.12 visitPre()

```
antlrcpp::Any CodeCheckVisitor::visitPre (
            ifccParser::PreContext * ctx )  [override], [virtual]
```

Visits a pre-unary operation (e.g., prefix increment/decrement).

This method checks for correctness in expressions involving pre-unary operations.

**Parameters**

| | |
|---|---|
| *ctx* | The context for the pre-unary expression. |

**Returns**

A result of the visit, typically unused.

Reimplemented from ifccBaseVisitor.

#### 5.8.2.13 visitReturn_stmt()

```
antlrcpp::Any CodeCheckVisitor::visitReturn_stmt (
            ifccParser::Return_stmtContext * ctx )  [override], [virtual]
```

Visits a return statement in the parsed code.

This method is used to process the return statement and check for any correctness issues, such as undefined variables.

**Parameters**

| | |
|---|---|
| *ctx* | The context for the return statement. |

**Returns**

A result of the visit, typically unused.

Reimplemented from ifccBaseVisitor.

#### 5.8.2.14 visitUnary()

```
antlrcpp::Any CodeCheckVisitor::visitUnary (
            ifccParser::UnaryContext * ctx )  [override], [virtual]
```

Visits a unary expression.

This method processes unary expressions (e.g., negation or logical NOT) and ensures that all variables in the expression are valid.

**Parameters**

| | |
|---|---|
| *ctx* | The context for the unary expression. |

**Returns**

A result of the visit, typically unused.

Reimplemented from ifccBaseVisitor.

### 5.8.3 Member Data Documentation

#### 5.8.3.1 currentOffset

```
int CodeCheckVisitor::currentOffset = 0  [protected]
```

The current offset value used for the symbols table.

#### 5.8.3.2 hasAValue

```
map<string, bool> CodeCheckVisitor::hasAValue  [protected]
```

A map to track whether a variable has been assigned a value.

#### 5.8.3.3 isUsed

```
map<string, bool> CodeCheckVisitor::isUsed  [protected]
```

A map indicating if a variable has been used in the code.

#### 5.8.3.4 symbolsTable

```
map<string, int> CodeCheckVisitor::symbolsTable  [protected]
```

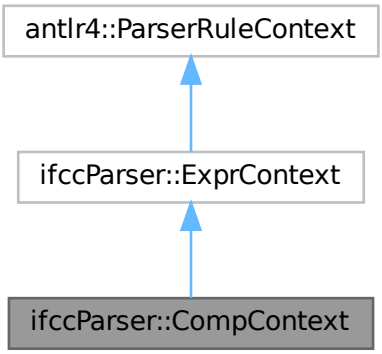
The symbols table containing variable names and their offsets.

The documentation for this class was generated from the following files:

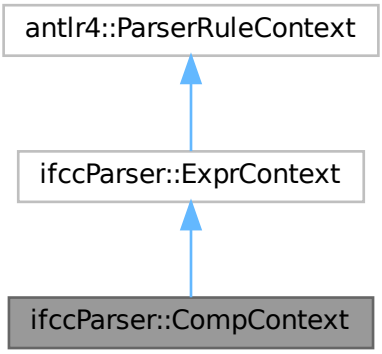
- CodeCheckVisitor.h
- CodeCheckVisitor.cpp

## 5.9 ifccParser::CompContext Class Reference

Inheritance diagram for ifccParser::CompContext:




Collaboration diagram for ifccParser::CompContext:



#### Public Member Functions

- **CompContext** (ExprContext ∗ctx)
- std::vector< ExprContext ∗ > **expr** ()
- ExprContext ∗ **expr** (size_t i)
- virtual std::any **accept** (antlr4::tree::ParseTreeVisitor ∗visitor) override

### Public Member Functions inherited from ifccParser::ExprContext

- **ExprContext** (antlr4::ParserRuleContext ∗parent, size_t invokingState)
- void **copyFrom** (ExprContext ∗context)
- virtual size_t **getRuleIndex** () const override

#### Public Attributes

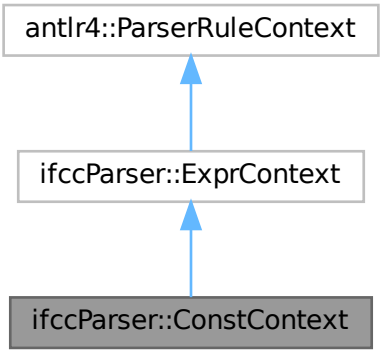
- antlr4::Token ∗ **OP** = nullptr

The documentation for this class was generated from the following files:

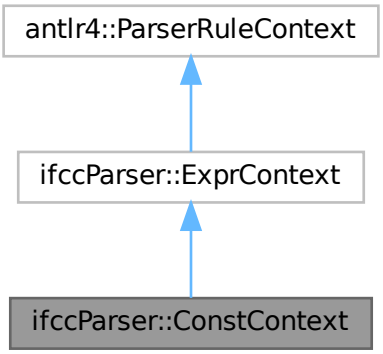
- ifccParser.h
- ifccParser.cpp

## 5.10 ifccParser::ConstContext Class Reference

Inheritance diagram for ifccParser::ConstContext:



Collaboration diagram for ifccParser::ConstContext:




**Public Member Functions**

- **ConstContext** (ExprContext ∗ctx)
- antlr4::tree::TerminalNode ∗ **CONST** ()
- virtual std::any **accept** (antlr4::tree::ParseTreeVisitor ∗visitor) override

### Public Member Functions inherited from ifccParser::ExprContext

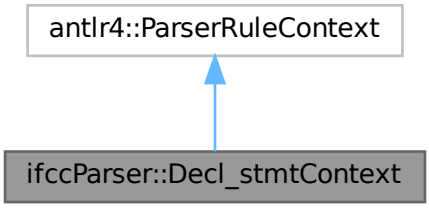
- **ExprContext** (antlr4::ParserRuleContext ∗parent, size_t invokingState)
- void **copyFrom** (ExprContext ∗context)
- virtual size_t **getRuleIndex** () const override

The documentation for this class was generated from the following files:

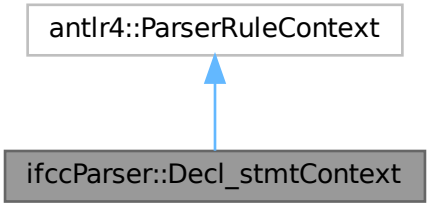
- ifccParser.h
- ifccParser.cpp

## 5.11 ifccParser::Decl_stmtContext Class Reference

Inheritance diagram for ifccParser::Decl_stmtContext:



Collaboration diagram for ifccParser::Decl_stmtContext:




### Public Member Functions

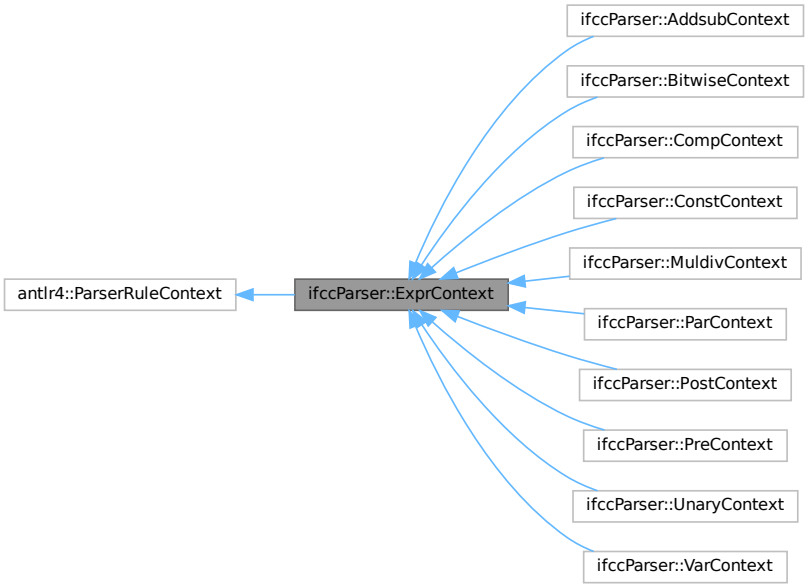
- **Decl_stmtContext** (antlr4::ParserRuleContext ∗parent, size_t invokingState)
- virtual size_t **getRuleIndex** () const override
- antlr4::tree::TerminalNode ∗ **TYPE** ()
- std::vector< antlr4::tree::TerminalNode ∗ > **VAR** ()
- antlr4::tree::TerminalNode ∗ **VAR** (size_t i)
- std::vector< ExprContext ∗ > **expr** ()
- ExprContext ∗ **expr** (size_t i)
- virtual std::any **accept** (antlr4::tree::ParseTreeVisitor ∗visitor) override

The documentation for this class was generated from the following files:

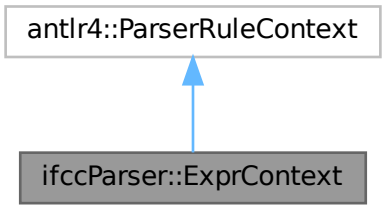
- ifccParser.h
- ifccParser.cpp

## 5.12 ifccParser::ExprContext Class Reference

Inheritance diagram for ifccParser::ExprContext:




Collaboration diagram for ifccParser::ExprContext:




### Public Member Functions

- **ExprContext** (antlr4::ParserRuleContext ∗parent, size_t invokingState)
- void **copyFrom** (ExprContext ∗context)
- virtual size_t **getRuleIndex** () const override

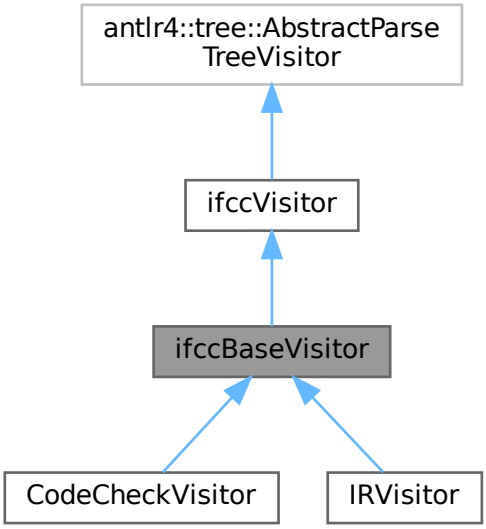
The documentation for this class was generated from the following files:

- ifccParser.h
- ifccParser.cpp

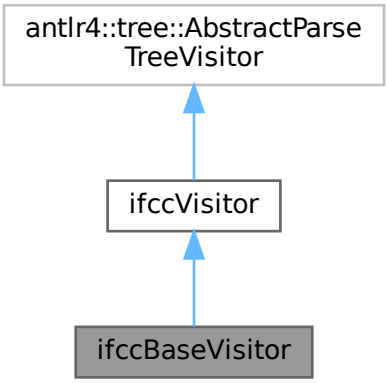
## 5.13 ifccBaseVisitor Class Reference

`#include <ifccBaseVisitor.h>`

Inheritance diagram for ifccBaseVisitor:




Collaboration diagram for ifccBaseVisitor:




### Public Member Functions

- virtual std::any visitAxiom (ifccParser::AxiomContext *ctx) override

- virtual std::any visitProg (ifccParser::ProgContext *ctx) override
- virtual std::any visitStatement (ifccParser::StatementContext *ctx) override
- virtual std::any visitDecl_stmt (ifccParser::Decl_stmtContext *ctx) override
- virtual std::any visitAssign_stmt (ifccParser::Assign_stmtContext *ctx) override
- virtual std::any visitIncrdecr_stmt (ifccParser::Incrdecr_stmtContext *ctx) override
- virtual std::any visitReturn_stmt (ifccParser::Return_stmtContext *ctx) override
- virtual std::any visitPar (ifccParser::ParContext *ctx) override
- virtual std::any visitComp (ifccParser::CompContext *ctx) override
- virtual std::any visitPre (ifccParser::PreContext *ctx) override
- virtual std::any visitConst (ifccParser::ConstContext *ctx) override
- virtual std::any visitPost (ifccParser::PostContext *ctx) override
- virtual std::any visitVar (ifccParser::VarContext *ctx) override
- virtual std::any visitBitwise (ifccParser::BitwiseContext *ctx) override
- virtual std::any visitAddsub (ifccParser::AddsubContext *ctx) override
- virtual std::any visitUnary (ifccParser::UnaryContext *ctx) override
- virtual std::any visitMuldiv (ifccParser::MuldivContext *ctx) override

## 5.13.1 Detailed Description

This class provides an empty implementation of ifccVisitor, which can be extended to create a visitor which only needs to handle a subset of the available methods.

## 5.13.2 Member Function Documentation

### 5.13.2.1 visitAddsub()

```
virtual std::any ifccBaseVisitor::visitAddsub (
            ifccParser::AddsubContext * ctx )  [inline], [override], [virtual]
```

Implements ifccVisitor.

Reimplemented in CodeCheckVisitor, and IRVisitor.

### 5.13.2.2 visitAssign_stmt()

```
virtual std::any ifccBaseVisitor::visitAssign_stmt (
            ifccParser::Assign_stmtContext * ctx )  [inline], [override], [virtual]
```

Implements ifccVisitor.

Reimplemented in CodeCheckVisitor, and IRVisitor.

### 5.13.2.3 visitAxiom()

```
virtual std::any ifccBaseVisitor::visitAxiom (
            ifccParser::AxiomContext * context )  [inline], [override], [virtual]
```

Visit parse trees produced by ifccParser.

Implements ifccVisitor.

#### 5.13.2.4 visitBitwise()

```
virtual std::any ifccBaseVisitor::visitBitwise (
            ifccParser::BitwiseContext * ctx )  [inline], [override], [virtual]
```

Implements ifccVisitor.

Reimplemented in CodeCheckVisitor, and IRVisitor.

#### 5.13.2.5 visitComp()

```
virtual std::any ifccBaseVisitor::visitComp (
            ifccParser::CompContext * ctx )  [inline], [override], [virtual]
```

Implements ifccVisitor.

Reimplemented in CodeCheckVisitor, and IRVisitor.

#### 5.13.2.6 visitConst()

```
virtual std::any ifccBaseVisitor::visitConst (
            ifccParser::ConstContext * ctx )  [inline], [override], [virtual]
```

Implements ifccVisitor.

#### 5.13.2.7 visitDecl_stmt()

```
virtual std::any ifccBaseVisitor::visitDecl_stmt (
            ifccParser::Decl_stmtContext * ctx )  [inline], [override], [virtual]
```

Implements ifccVisitor.

Reimplemented in CodeCheckVisitor, and IRVisitor.

#### 5.13.2.8 visitIncrdecr_stmt()

```
virtual std::any ifccBaseVisitor::visitIncrdecr_stmt (
            ifccParser::Incrdecr_stmtContext * ctx )  [inline], [override], [virtual]
```

Implements ifccVisitor.

#### 5.13.2.9 visitMuldiv()

```
virtual std::any ifccBaseVisitor::visitMuldiv (
            ifccParser::MuldivContext * ctx )  [inline], [override], [virtual]
```

Implements ifccVisitor.

Reimplemented in CodeCheckVisitor, and IRVisitor.

#### 5.13.2.10 visitPar()

```
virtual std::any ifccBaseVisitor::visitPar (
            ifccParser::ParContext * ctx )  [inline], [override], [virtual]
```

Implements ifccVisitor.

#### 5.13.2.11 visitPost()

```
virtual std::any ifccBaseVisitor::visitPost (
            ifccParser::PostContext * ctx )  [inline], [override], [virtual]
```

Implements ifccVisitor.

Reimplemented in CodeCheckVisitor, and IRVisitor.

#### 5.13.2.12 visitPre()

```
virtual std::any ifccBaseVisitor::visitPre (
            ifccParser::PreContext * ctx )  [inline], [override], [virtual]
```

Implements ifccVisitor.

Reimplemented in CodeCheckVisitor, and IRVisitor.

#### 5.13.2.13 visitProg()

```
virtual std::any ifccBaseVisitor::visitProg (
            ifccParser::ProgContext * ctx )  [inline], [override], [virtual]
```

Implements ifccVisitor.

Reimplemented in IRVisitor.

#### 5.13.2.14 visitReturn_stmt()

```
virtual std::any ifccBaseVisitor::visitReturn_stmt (
            ifccParser::Return_stmtContext * ctx )  [inline], [override], [virtual]
```

Implements ifccVisitor.

Reimplemented in CodeCheckVisitor, and IRVisitor.

#### 5.13.2.15 visitStatement()

```
virtual std::any ifccBaseVisitor::visitStatement (
            ifccParser::StatementContext * ctx )  [inline], [override], [virtual]
```

Implements ifccVisitor.

#### 5.13.2.16 visitUnary()

```
virtual std::any ifccBaseVisitor::visitUnary (
            ifccParser::UnaryContext * ctx )  [inline], [override], [virtual]
```

Implements ifccVisitor.

Reimplemented in CodeCheckVisitor, and IRVisitor.

#### 5.13.2.17 visitVar()

```
virtual std::any ifccBaseVisitor::visitVar (
            ifccParser::VarContext * ctx )  [inline], [override], [virtual]
```

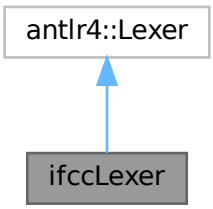
Implements ifccVisitor.

The documentation for this class was generated from the following file:

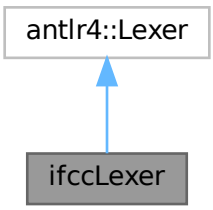
- ifccBaseVisitor.h

## 5.14 ifccLexer Class Reference

Inheritance diagram for ifccLexer:




Collaboration diagram for ifccLexer:



#### Public Types

- enum {
  **T__0** = 1 , **T__1** = 2 , **T__2** = 3 , **T__3** = 4 ,
  **T__4** = 5 , **T__5** = 6 , **T__6** = 7 , **T__7** = 8 ,
  **T__8** = 9 , **T__9** = 10 , **T__10** = 11 , **T__11** = 12 ,
  **T__12** = 13 , **T__13** = 14 , **T__14** = 15 , **T__15** = 16 ,
  **T__16** = 17 , **T__17** = 18 , **T__18** = 19 , **T__19** = 20 ,
  **T__20** = 21 , **T__21** = 22 , **T__22** = 23 , **T__23** = 24 ,
  **T__24** = 25 , **T__25** = 26 , **OPU** = 27 , **RETURN** = 28 ,
  **TYPE** = 29 , **VAR** = 30 , **CONST** = 31 , **COMMENT** = 32 ,
  **DIRECTIVE** = 33 , **WS** = 34 }

#### Public Member Functions

- **ifccLexer** (antlr4::CharStream ∗input)
- std::string **getGrammarFileName** () const override
- const std::vector< std::string > & **getRuleNames** () const override
- const std::vector< std::string > & **getChannelNames** () const override
- const std::vector< std::string > & **getModeNames** () const override
- const antlr4::dfa::Vocabulary & **getVocabulary** () const override
- antlr4::atn::SerializedATNView **getSerializedATN** () const override
- const antlr4::atn::ATN & **getATN** () const override

#### Static Public Member Functions

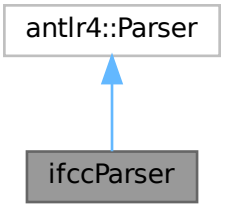
- static void **initialize** ()

The documentation for this class was generated from the following files:

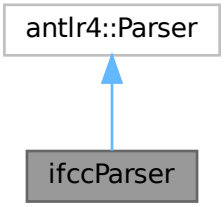
- ifccLexer.h
- ifccLexer.cpp

## 5.15 ifccParser Class Reference

Inheritance diagram for ifccParser:



Collaboration diagram for ifccParser:




### Classes

- class AddsubContext
- class Assign_stmtContext
- class AxiomContext
- class BitwiseContext
- class CompContext
- class ConstContext
- class Decl_stmtContext
- class ExprContext
- class Incrdecr_stmtContext
- class MuldivContext
- class ParContext
- class PostContext
- class PreContext
- class ProgContext
- class Return_stmtContext
- class StatementContext
- class UnaryContext
- class VarContext

### Public Types

- enum {
  **T__0** = 1 , **T__1** = 2 , **T__2** = 3 , **T__3** = 4 ,
  **T__4** = 5 , **T__5** = 6 , **T__6** = 7 , **T__7** = 8 ,
  **T__8** = 9 , **T__9** = 10 , **T__10** = 11 , **T__11** = 12 ,
  **T__12** = 13 , **T__13** = 14 , **T__14** = 15 , **T__15** = 16 ,
  **T__16** = 17 , **T__17** = 18 , **T__18** = 19 , **T__19** = 20 ,
  **T__20** = 21 , **T__21** = 22 , **T__22** = 23 , **T__23** = 24 ,
  **T__24** = 25 , **T__25** = 26 , **OPU** = 27 , **RETURN** = 28 ,
  **TYPE** = 29 , **VAR** = 30 , **CONST** = 31 , **COMMENT** = 32 ,
  **DIRECTIVE** = 33 , **WS** = 34 }
- enum {
  **RuleAxiom** = 0 , **RuleProg** = 1 , **RuleStatement** = 2 , **RuleDecl_stmt** = 3 ,
  **RuleAssign_stmt** = 4 , **RuleIncrdecr_stmt** = 5 , **RuleReturn_stmt** = 6 , **RuleExpr** = 7 }

### Public Member Functions

- **ifccParser** (antlr4::TokenStream ∗input)
- **ifccParser** (antlr4::TokenStream ∗input, const antlr4::atn::ParserATNSimulatorOptions &options)
- std::string **getGrammarFileName** () const override
- const antlr4::atn::ATN & **getATN** () const override
- const std::vector< std::string > & **getRuleNames** () const override
- const antlr4::dfa::Vocabulary & **getVocabulary** () const override
- antlr4::atn::SerializedATNView **getSerializedATN** () const override
- AxiomContext ∗ **axiom** ()
- ProgContext ∗ **prog** ()
- StatementContext ∗ **statement** ()
- Decl_stmtContext ∗ **decl_stmt** ()
- Assign_stmtContext ∗ **assign_stmt** ()
- Incrdecr_stmtContext ∗ **incrdecr_stmt** ()
- Return_stmtContext ∗ **return_stmt** ()
- ExprContext ∗ **expr** ()
- ExprContext ∗ **expr** (int precedence)
- bool **sempred** (antlr4::RuleContext ∗_localctx, size_t ruleIndex, size_t predicateIndex) override
- bool **exprSempred** (ExprContext ∗_localctx, size_t predicateIndex)

### Static Public Member Functions

- static void **initialize** ()

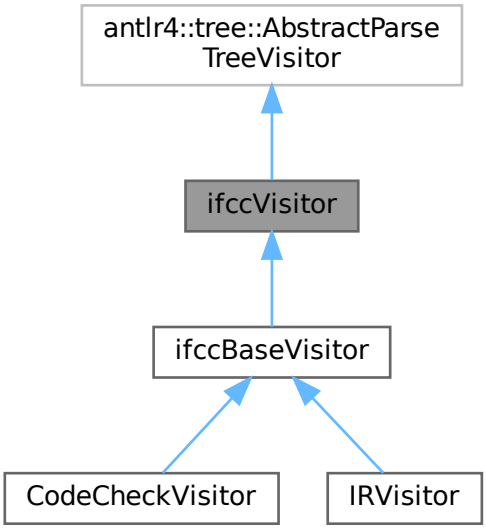
The documentation for this class was generated from the following files:

- ifccParser.h
- ifccParser.cpp

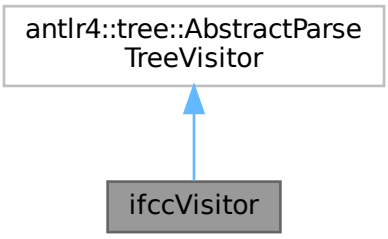
## 5.16 ifccVisitor Class Reference

`#include <ifccVisitor.h>`

Inheritance diagram for ifccVisitor:




Collaboration diagram for ifccVisitor:




### Public Member Functions

- virtual std::any visitAxiom (ifccParser::AxiomContext ∗context)=0
- virtual std::any visitProg (ifccParser::ProgContext ∗context)=0
- virtual std::any **visitStatement** (ifccParser::StatementContext ∗context)=0
- virtual std::any visitDecl_stmt (ifccParser::Decl_stmtContext ∗context)=0
- virtual std::any visitAssign_stmt (ifccParser::Assign_stmtContext ∗context)=0
- virtual std::any **visitIncrdecr_stmt** (ifccParser::Incrdecr_stmtContext ∗context)=0
- virtual std::any visitReturn_stmt (ifccParser::Return_stmtContext ∗context)=0
- virtual std::any **visitPar** (ifccParser::ParContext ∗context)=0
- virtual std::any visitComp (ifccParser::CompContext ∗context)=0

- virtual std::any visitPre (ifccParser::PreContext ∗context)=0
- virtual std::any **visitConst** (ifccParser::ConstContext ∗context)=0
- virtual std::any visitPost (ifccParser::PostContext ∗context)=0
- virtual std::any **visitVar** (ifccParser::VarContext ∗context)=0
- virtual std::any visitBitwise (ifccParser::BitwiseContext ∗context)=0
- virtual std::any visitAddsub (ifccParser::AddsubContext ∗context)=0
- virtual std::any visitUnary (ifccParser::UnaryContext ∗context)=0
- virtual std::any visitMuldiv (ifccParser::MuldivContext ∗context)=0

## 5.16.1 Detailed Description

This class defines an abstract visitor for a parse tree produced by ifccParser.

## 5.16.2 Member Function Documentation

### 5.16.2.1 visitAddsub()

```
virtual std::any ifccVisitor::visitAddsub (
            ifccParser::AddsubContext * context )  [pure virtual]
```

Implemented in CodeCheckVisitor, and IRVisitor.

### 5.16.2.2 visitAssign_stmt()

```
virtual std::any ifccVisitor::visitAssign_stmt (
            ifccParser::Assign_stmtContext * context )  [pure virtual]
```

Implemented in CodeCheckVisitor, and IRVisitor.

### 5.16.2.3 visitAxiom()

```
virtual std::any ifccVisitor::visitAxiom (
            ifccParser::AxiomContext * context )  [pure virtual]
```

Visit parse trees produced by ifccParser.

Implemented in ifccBaseVisitor.

### 5.16.2.4 visitBitwise()

```
virtual std::any ifccVisitor::visitBitwise (
            ifccParser::BitwiseContext * context )  [pure virtual]
```

Implemented in CodeCheckVisitor, and IRVisitor.

#### 5.16.2.5 visitComp()

```
virtual std::any ifccVisitor::visitComp (
            ifccParser::CompContext * context )  [pure virtual]
```

Implemented in CodeCheckVisitor, and IRVisitor.

#### 5.16.2.6 visitDecl_stmt()

```
virtual std::any ifccVisitor::visitDecl_stmt (
            ifccParser::Decl_stmtContext * context )  [pure virtual]
```

Implemented in CodeCheckVisitor, and IRVisitor.

#### 5.16.2.7 visitMuldiv()

```
virtual std::any ifccVisitor::visitMuldiv (
            ifccParser::MuldivContext * context )  [pure virtual]
```

Implemented in CodeCheckVisitor, and IRVisitor.

#### 5.16.2.8 visitPost()

```
virtual std::any ifccVisitor::visitPost (
            ifccParser::PostContext * context )  [pure virtual]
```

Implemented in CodeCheckVisitor, and IRVisitor.

#### 5.16.2.9 visitPre()

```
virtual std::any ifccVisitor::visitPre (
            ifccParser::PreContext * context )  [pure virtual]
```

Implemented in CodeCheckVisitor, and IRVisitor.

#### 5.16.2.10 visitProg()

```
virtual std::any ifccVisitor::visitProg (
            ifccParser::ProgContext * context )  [pure virtual]
```

Implemented in IRVisitor.

#### 5.16.2.11 visitReturn_stmt()

```
virtual std::any ifccVisitor::visitReturn_stmt (
            ifccParser::Return_stmtContext * context )  [pure virtual]
```

Implemented in CodeCheckVisitor, and IRVisitor.

#### 5.16.2.12 visitUnary()

```
virtual std::any ifccVisitor::visitUnary (
            ifccParser::UnaryContext * context )  [pure virtual]
```

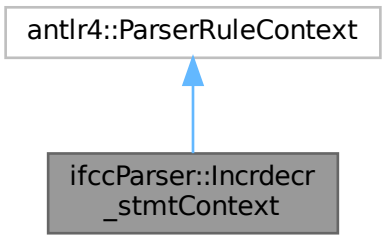
Implemented in CodeCheckVisitor, and IRVisitor.

The documentation for this class was generated from the following file:

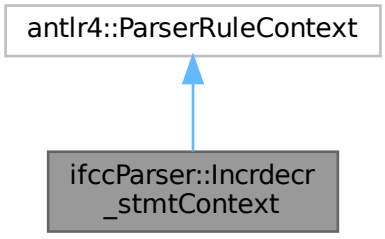
- ifccVisitor.h

## 5.17 ifccParser::Incrdecr_stmtContext Class Reference

Inheritance diagram for ifccParser::Incrdecr_stmtContext:




Collaboration diagram for ifccParser::Incrdecr_stmtContext:




### Public Member Functions

- **Incrdecr_stmtContext** (antlr4::ParserRuleContext ∗parent, size_t invokingState)
- virtual size_t **getRuleIndex** () const override
- antlr4::tree::TerminalNode ∗ **VAR** ()
- virtual std::any **accept** (antlr4::tree::ParseTreeVisitor ∗visitor) override

**Public Attributes**

- antlr4::Token * **OP** = nullptr

The documentation for this class was generated from the following files:

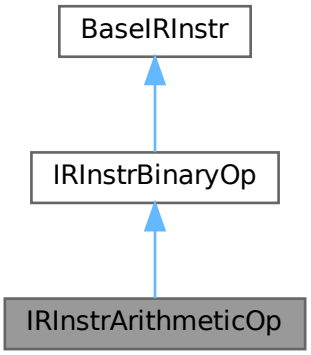
- ifccParser.h
- ifccParser.cpp

## 5.18 IRInstrArithmeticOp Class Reference

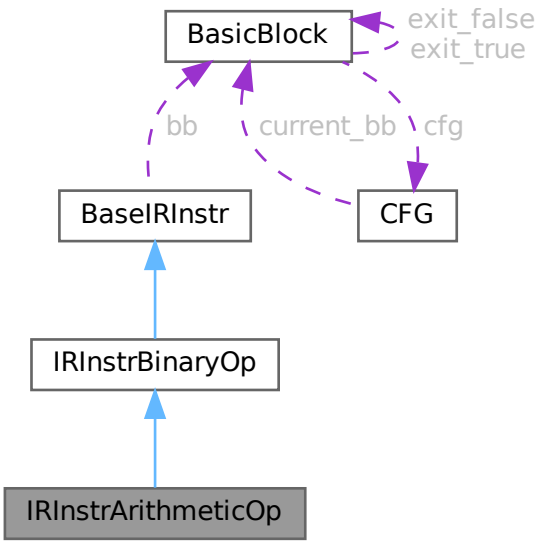
Represents an arithmetic operation instruction in the intermediate representation.

`#include <IRInstrArithmeticOp.h>`

Inheritance diagram for IRInstrArithmeticOp:



Collaboration diagram for IRInstrArithmeticOp:




### Public Member Functions

- IRInstrArithmeticOp (BasicBlock *bb_, string firstOp, string secondOp, string op)
  *Constructs an arithmetic operation instruction.*
- virtual void gen_asm (ostream &o)
  *Generates the assembly code for this arithmetic operation instruction.*

## Public Member Functions inherited from IRInstrBinaryOp

- IRInstrBinaryOp (BasicBlock *bb_, string firstOp, string secondOp, string op)
  *Constructs a binary operation instruction.*

## Public Member Functions inherited from BaseIRInstr

- BaseIRInstr (BasicBlock *bb_)
  *Constructs an instruction for a given basic block.*
- BasicBlock * getBB ()
  *Gets the basic block that this instruction belongs to.*

### Additional Inherited Members

## Protected Attributes inherited from IRInstrBinaryOp

- string firstOp
  *The first operand for the binary operation.*
- string secondOp
  *The second operand for the binary operation.*
- string op
  *The binary operation (e.g., '+', '-', '*', '/',").*

### Protected Attributes inherited from BaseIRInstr

- BasicBlock * bb

  *The basic block that this instruction belongs to.*

## 5.18.1 Detailed Description

Represents an arithmetic operation instruction in the intermediate representation.

This class handles the generation of intermediate representation instructions for arithmetic operations, such as addition, subtraction, multiplication, division, modulo, and bitwise operations. It extends the IRInstrBinaryOp class and provides specialized methods for handling these operations.

## 5.18.2 Constructor & Destructor Documentation

### 5.18.2.1 IRInstrArithmeticOp()

```
IRInstrArithmeticOp::IRInstrArithmeticOp (
            BasicBlock * bb_,
            string firstOp,
            string secondOp,
            string op )  [inline]
```

Constructs an arithmetic operation instruction.

Initializes the instruction with a basic block, two operands, and the arithmetic operation.

**Parameters**

| | |
|---|---|
| *bb_* | The basic block to which the instruction belongs. |
| *firstOp* | The first operand of the arithmetic operation. |
| *secondOp* | The second operand of the arithmetic operation. |
| *op* | The arithmetic operation (e.g., '+', '-', '*', '/', ''). |

## 5.18.3 Member Function Documentation

### 5.18.3.1 gen_asm()

```
void IRInstrArithmeticOp::gen_asm (
            ostream & o )  [virtual]
```

Generates the assembly code for this arithmetic operation instruction.

This method generates the appropriate assembly code based on the specific arithmetic operation (e.g., addition, subtraction, multiplication, division, modulo, or bitwise operations).

**Parameters**

| | |
|---|---|
| *o* | The output stream where the generated assembly code will be written. |

Implements IRInstrBinaryOp.

The documentation for this class was generated from the following files:

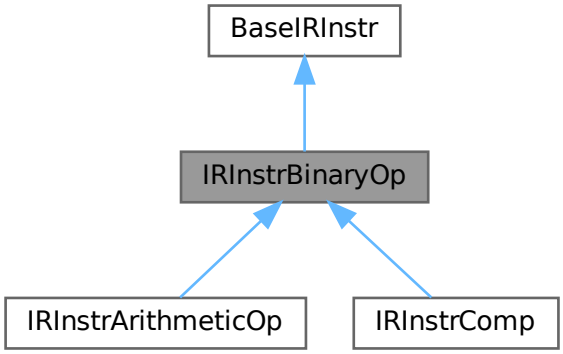
- IRInstrArithmeticOp.h
- IRInstrArithmeticOp.cpp

## 5.19 IRInstrBinaryOp Class Reference

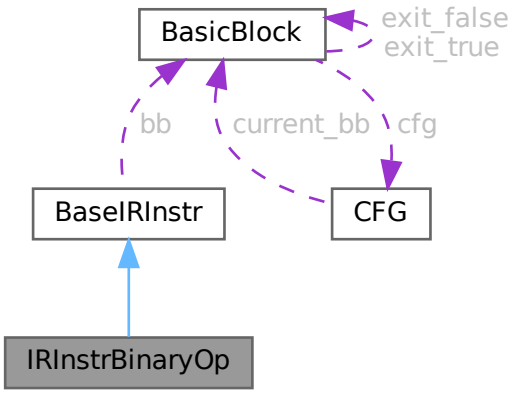
Represents a binary operation instruction in the intermediate representation.

`#include <IRInstrBinaryOp.h>`

Inheritance diagram for IRInstrBinaryOp:




Collaboration diagram for IRInstrBinaryOp:



#### Public Member Functions

- IRInstrBinaryOp (BasicBlock *bb_, string firstOp, string secondOp, string op)

  *Constructs a binary operation instruction.*
- virtual void gen_asm (ostream &o)=0

  *Generates the assembly code for this binary operation instruction.*

### Public Member Functions inherited from BaseIRInstr

- BaseIRInstr (BasicBlock *bb_)

  *Constructs an instruction for a given basic block.*
- BasicBlock * getBB ()

  *Gets the basic block that this instruction belongs to.*

#### Protected Attributes

- string firstOp

  *The first operand for the binary operation.*
- string secondOp

  *The second operand for the binary operation.*
- string op

  *The binary operation (e.g., '+', '-', '*', '/',").*

### Protected Attributes inherited from BaseIRInstr

- BasicBlock * bb

  *The basic block that this instruction belongs to.*

## 5.19.1 Detailed Description

Represents a binary operation instruction in the intermediate representation.

This class serves as the base class for binary operation instructions such as addition, subtraction, multiplication, etc. It provides a structure for managing two operands and the operation itself, and it also handles the generation of the corresponding assembly code for binary operations.

## 5.19.2 Constructor & Destructor Documentation

### 5.19.2.1 IRInstrBinaryOp()

```
IRInstrBinaryOp::IRInstrBinaryOp (
            BasicBlock * bb_,
            string firstOp,
            string secondOp,
            string op )  [inline]
```

Constructs a binary operation instruction.

Initializes the instruction with a basic block, two operands, and the binary operation.

**Parameters**

| | |
|---|---|
| *bb_* | The basic block to which the instruction belongs. |
| *firstOp* | The first operand of the binary operation. |
| *secondOp* | The second operand of the binary operation. |
| *op* | The binary operation (e.g., '+', '-', '*', '/'). |

## 5.19.3 Member Function Documentation

### 5.19.3.1 gen_asm()

```
virtual void IRInstrBinaryOp::gen_asm (
            ostream & o )  [pure virtual]
```

Generates the assembly code for this binary operation instruction.

This method must be implemented by derived classes to generate the appropriate assembly code based on the specific binary operation.

**Parameters**

| | |
|---|---|
| *o* | The output stream where the generated assembly code will be written. |

Implements BaseIRInstr.

Implemented in IRInstrArithmeticOp, and IRInstrComp.

## 5.19.4 Member Data Documentation

### 5.19.4.1 firstOp

```
string IRInstrBinaryOp::firstOp  [protected]
```

The first operand for the binary operation.

### 5.19.4.2 op

```
string IRInstrBinaryOp::op  [protected]
```

The binary operation (e.g., '+', '-', '*', '/','').

### 5.19.4.3 secondOp

```
string IRInstrBinaryOp::secondOp  [protected]
```

The second operand for the binary operation.

The documentation for this class was generated from the following file:

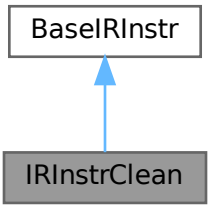
- IRInstrBinaryOp.h

## 5.20 IRInstrClean Class Reference

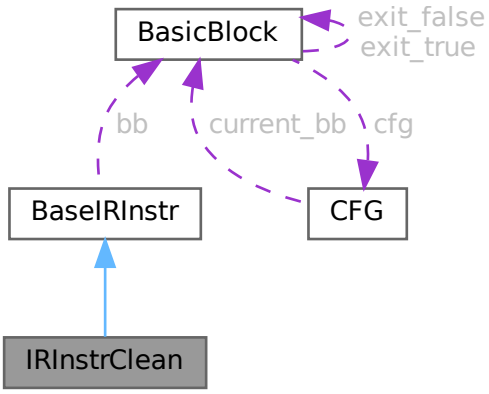
Represents a clean-up instruction in the intermediate representation.

`#include <IRInstrClean.h>`

Inheritance diagram for IRInstrClean:




Collaboration diagram for IRInstrClean:




### Public Member Functions

- IRInstrClean (BasicBlock *bb_)

  *Constructs an IRInstrClean object.*
- virtual void gen_asm (std::ostream &o) override

  *Generates assembly code for the IRInstrClean instruction.*

## Public Member Functions inherited from BaseIRInstr

- BaseIRInstr (BasicBlock *bb_)

  *Constructs an instruction for a given basic block.*
- BasicBlock * getBB ()

  *Gets the basic block that this instruction belongs to.*
- virtual void gen_asm (ostream &o)=0

  *Generates the assembly code for this instruction.*

### Additional Inherited Members

## Protected Attributes inherited from BaseIRInstr

- BasicBlock * bb

  *The basic block that this instruction belongs to.*

## 5.20.1 Detailed Description

Represents a clean-up instruction in the intermediate representation.

The `IRInstrClean` class is a subclass of `BaseIRInstr`. It represents a clean-up instruction, typically used to deallocate resources or reset values within a basic block during intermediate representation generation. The main responsibility of this class is to generate the corresponding assembly code for the clean-up operation.

## 5.20.2 Constructor & Destructor Documentation

### 5.20.2.1 IRInstrClean()

```
IRInstrClean::IRInstrClean (
            BasicBlock * bb_ )  [inline]
```

Constructs an IRInstrClean object.

This constructor initializes an `IRInstrClean` instance with a reference to the basic block where this clean-up instruction resides.

**Parameters**

| | |
|---|---|
| *bb_* | A pointer to the `BasicBlock` to which this instruction belongs. |

## 5.20.3 Member Function Documentation

### 5.20.3.1 gen_asm()

```
void IRInstrClean::gen_asm (
            std::ostream & o )  [override], [virtual]
```

Generates assembly code for the IRInstrClean instruction.

This function generates the assembly code corresponding to the clean-up operation represented by this instruction and writes it to the provided output stream.

The generated assembly code typically involves operations to reset, deallocate, or clean up resources associated with the instruction.

**Parameters**

| | |
|---|---|
| *o* | The output stream to which the generated assembly code will be written. |

The documentation for this class was generated from the following files:

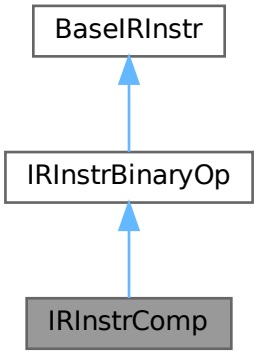
- IRInstrClean.h
- IRInstrClean.cpp

## 5.21 IRInstrComp Class Reference

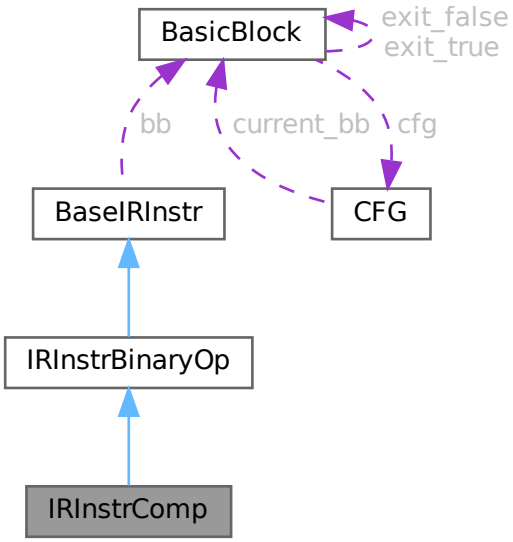
Represents a comparison operation instruction in the intermediate representation.

`#include <IRInstrComp.h>`

Inheritance diagram for IRInstrComp:



Collaboration diagram for IRInstrComp:




### Public Member Functions

- IRInstrComp (BasicBlock *bb_, string firstOp, string secondOp, string op)
  *Constructs a comparison operation instruction.*
- virtual void gen_asm (ostream &o) override
  *Generates the assembly code for this comparison operation instruction.*

## Public Member Functions inherited from IRInstrBinaryOp

- IRInstrBinaryOp (BasicBlock *bb_, string firstOp, string secondOp, string op)
  *Constructs a binary operation instruction.*

## Public Member Functions inherited from BaseIRInstr

- BaseIRInstr (BasicBlock *bb_)
  *Constructs an instruction for a given basic block.*
- BasicBlock * getBB ()
  *Gets the basic block that this instruction belongs to.*

### Additional Inherited Members

## Protected Attributes inherited from IRInstrBinaryOp

- string firstOp
  *The first operand for the binary operation.*
- string secondOp
  *The second operand for the binary operation.*
- string op
  *The binary operation (e.g., '+', '-', '*', '/',").*

### Protected Attributes inherited from BaseIRInstr

- BasicBlock * bb

  *The basic block that this instruction belongs to.*

## 5.21.1 Detailed Description

Represents a comparison operation instruction in the intermediate representation.

This class handles the generation of intermediate representation instructions for comparison operations, such as equality, inequality, greater than, greater than or equal to, less than, and less than or equal to.

## 5.21.2 Constructor & Destructor Documentation

### 5.21.2.1 IRInstrComp()

```
IRInstrComp::IRInstrComp (
            BasicBlock * bb_,
            string firstOp,
            string secondOp,
            string op )  [inline]
```

Constructs a comparison operation instruction.

Initializes the instruction with a basic block, two operands, and the comparison operation.

**Parameters**

| | |
|---|---|
| *bb_* | The basic block to which the instruction belongs. |
| *firstOp* | The first operand of the comparison operation. |
| *secondOp* | The second operand of the comparison operation. |
| *op* | The comparison operation (e.g., '==', '!=', '>', '<', '>=', '<='). |

## 5.21.3 Member Function Documentation

### 5.21.3.1 gen_asm()

```
void IRInstrComp::gen_asm (
            ostream & o )  [override], [virtual]
```

Generates the assembly code for this comparison operation instruction.

This method generates the appropriate assembly code based on the specific comparison operation (e.g., equality, inequality, greater than, greater than or equal to, less than, or less than or equal to).

**Parameters**

| | |
|---|---|
| *o* | The output stream where the generated assembly code will be written. |

Implements IRInstrBinaryOp.

The documentation for this class was generated from the following files:

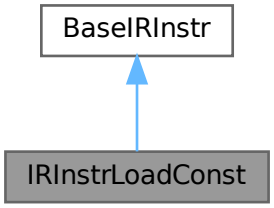
- IRInstrComp.h
- IRInstrComp.cpp

## 5.22 IRInstrLoadConst Class Reference

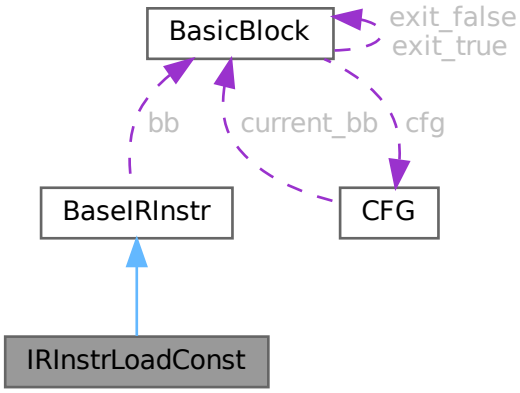
Represents an IR instruction for loading a constant into memory or a register.

`#include <IRInstrLoadConst.h>`

Inheritance diagram for IRInstrLoadConst:




Collaboration diagram for IRInstrLoadConst:



#### Public Member Functions

- IRInstrLoadConst (BasicBlock *bb_, int value, string dest)

  *Constructor for the IRInstrLoadConst instruction.*
- virtual void gen_asm (ostream &o) override

  *Generates assembly code to load a constant.*

### Public Member Functions inherited from BaseIRInstr

- BaseIRInstr (BasicBlock *bb_)

  *Constructs an instruction for a given basic block.*
- BasicBlock * getBB ()

  *Gets the basic block that this instruction belongs to.*

#### Additional Inherited Members

### Protected Attributes inherited from BaseIRInstr

- BasicBlock * bb

  *The basic block that this instruction belongs to.*

## 5.22.1 Detailed Description

Represents an IR instruction for loading a constant into memory or a register.

This instruction is used to assign an immediate value to a register or memory location.

## 5.22.2 Constructor & Destructor Documentation

### 5.22.2.1 IRInstrLoadConst()

```
IRInstrLoadConst::IRInstrLoadConst (
            BasicBlock * bb_,
            int value,
            string dest )
```

Constructor for the IRInstrLoadConst instruction.

Initializes the instruction with the basic block, the constant value to load, and the destination register or memory variable where the value should be stored.

**Parameters**

| | |
|---|---|
| *bb_* | Pointer to the basic block containing this instruction. |
| *value* | The constant value to load. |
| *dest* | The name of the target register or memory variable. |

### 5.22.3 Member Function Documentation

#### 5.22.3.1 gen_asm()

```
void IRInstrLoadConst::gen_asm (
            ostream & o )  [override], [virtual]
```

Generates assembly code to load a constant.

Generates assembly code to load a constant into memory or a register.

Generates assembly code for loading a constant value into a register or memory.

**Parameters**

| | |
|---|---|
| *o* | Output stream where the assembly code will be written. |

Implements BaseIRInstr.

The documentation for this class was generated from the following files:

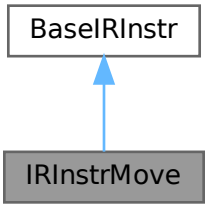
- IRInstrLoadConst.h
- IRInstrLoadConst.cpp

## 5.23 IRInstrMove Class Reference

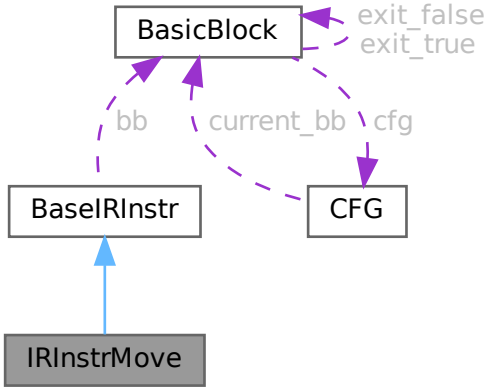
Represents an IR instruction for moving a value between registers and memory.

```
#include <IRInstrMove.h>
```

Inheritance diagram for IRInstrMove:



Collaboration diagram for IRInstrMove:




#### Public Member Functions

- IRInstrMove (BasicBlock ∗bb_, string src, string dest)

  *Constructor for the IRInstrMove instruction.*
- virtual void gen_asm (ostream &o) override

  *Generates the assembly code corresponding to the move instruction.*

### Public Member Functions inherited from BaseIRInstr

- BaseIRInstr (BasicBlock ∗bb_)

  *Constructs an instruction for a given basic block.*
- BasicBlock ∗ getBB ()

  *Gets the basic block that this instruction belongs to.*

#### Additional Inherited Members

### Protected Attributes inherited from BaseIRInstr

- BasicBlock ∗ bb

  *The basic block that this instruction belongs to.*

## 5.23.1 Detailed Description

Represents an IR instruction for moving a value between registers and memory.

This class handles data transfers between registers and the stack but does not manage constants.

### 5.23.2 Constructor & Destructor Documentation

#### 5.23.2.1 IRInstrMove()

```
IRInstrMove::IRInstrMove (
            BasicBlock * bb_,
            string src,
            string dest )
```

Constructor for the IRInstrMove instruction.

Initializes the instruction with the basic block, source, and destination variables.

**Parameters**

| | |
|---|---|
| *bb_* | Pointer to the basic block containing this instruction. |
| *src* | The name of the source variable (register or memory location). |
| *dest* | The name of the destination variable (register or memory location). |

### 5.23.3 Member Function Documentation

#### 5.23.3.1 gen_asm()

```
void IRInstrMove::gen_asm (
            ostream & o )  [override], [virtual]
```

Generates the assembly code corresponding to the move instruction.

Generates assembly code for moving a value between registers and memory.

This method generates the appropriate assembly code for moving a value from the source variable to the destination variable.

**Parameters**

| | |
|---|---|
| *o* | Output stream where the assembly code will be written. |

Implements BaseIRInstr.

The documentation for this class was generated from the following files:

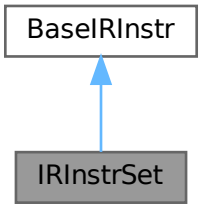
- IRInstrMove.h
- IRInstrMove.cpp

## 5.24 IRInstrSet Class Reference

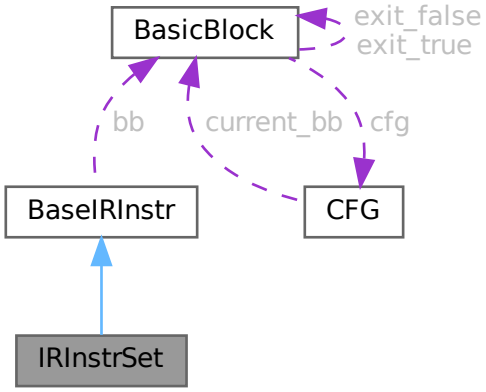
Represents an instruction that sets a value in the intermediate representation.

`#include <IRInstrSet.h>`

Inheritance diagram for IRInstrSet:




Collaboration diagram for IRInstrSet:




### Public Member Functions

- IRInstrSet (BasicBlock *bb_)

  *Constructs an IRInstrSet object.*
- virtual void gen_asm (std::ostream &o) override

  *Generates assembly code for the IRInstrSet instruction.*

## Public Member Functions inherited from BaseIRInstr

- BaseIRInstr (BasicBlock *bb_)

  *Constructs an instruction for a given basic block.*
- BasicBlock * getBB ()

  *Gets the basic block that this instruction belongs to.*
- virtual void gen_asm (ostream &o)=0

  *Generates the assembly code for this instruction.*

**Additional Inherited Members**

## Protected Attributes inherited from BaseIRInstr

- BasicBlock * bb

  *The basic block that this instruction belongs to.*

### 5.24.1 Detailed Description

Represents an instruction that sets a value in the intermediate representation.

The `IRInstrSet` class is a subclass of `BaseIRInstr`. It represents an instruction that performs an assignment or setting of a value within a basic block during intermediate representation generation. The primary function of this class is to generate the assembly code for the instruction.

### 5.24.2 Constructor & Destructor Documentation

#### 5.24.2.1 IRInstrSet()

```
IRInstrSet::IRInstrSet (
            BasicBlock * bb_ )  [inline]
```

Constructs an IRInstrSet object.

This constructor initializes an `IRInstrSet` instance with a reference to the basic block in which this instruction resides.

**Parameters**

| | |
|---|---|
| *bb_* | A pointer to the `BasicBlock` to which this instruction belongs. |

### 5.24.3 Member Function Documentation

#### 5.24.3.1 gen_asm()

```
void IRInstrSet::gen_asm (
            std::ostream & o )  [override], [virtual]
```

Generates assembly code for the IRInstrSet instruction.

This function generates the corresponding assembly code for the `IRInstrSet` instruction and writes it to the provided output stream.

The generated assembly code typically includes an instruction for setting a value in a register or memory location, depending on the context.

**Parameters**

| | |
|---|---|
| *o* | The output stream to which the generated assembly code will be written. |

The documentation for this class was generated from the following files:

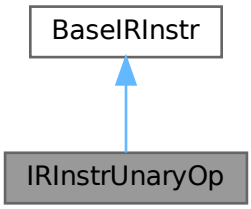
- IRInstrSet.h
- IRInstrSet.cpp

## 5.25 IRInstrUnaryOp Class Reference

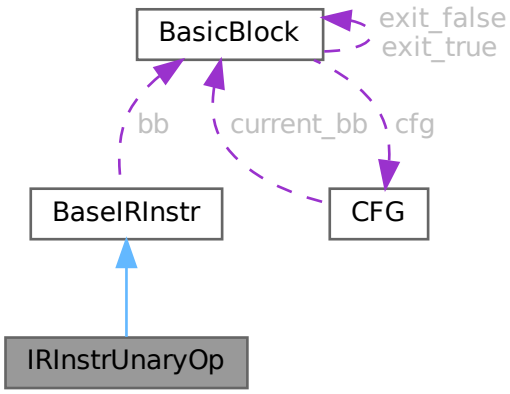
Represents a unary operation instruction in the intermediate representation.

`#include <IRInstrUnaryOp.h>`

Inheritance diagram for IRInstrUnaryOp:




Collaboration diagram for IRInstrUnaryOp:



#### Public Member Functions

- IRInstrUnaryOp (BasicBlock *bb_, string uniqueOp, string op)

  *Constructs a unary operation instruction.*
- virtual void gen_asm (ostream &o)

  *Generates the assembly code for this unary operation instruction.*

### Public Member Functions inherited from BaseIRInstr

- BaseIRInstr (BasicBlock *bb_)

  *Constructs an instruction for a given basic block.*
- BasicBlock * getBB ()

  *Gets the basic block that this instruction belongs to.*

#### Protected Attributes

- string uniqueOp

  *A unique identifier for the unary operation.*
- string op

  *The actual unary operation (e.g., '!', '-', '~').*

### Protected Attributes inherited from BaseIRInstr

- BasicBlock * bb

  *The basic block that this instruction belongs to.*

## 5.25.1 Detailed Description

Represents a unary operation instruction in the intermediate representation.

This class handles the generation of intermediate representation instructions for unary operations, such as negation, logical negation, and bitwise complement.

## 5.25.2 Constructor & Destructor Documentation

### 5.25.2.1 IRInstrUnaryOp()

```
IRInstrUnaryOp::IRInstrUnaryOp (
            BasicBlock * bb_,
            string uniqueOp,
            string op )  [inline]
```

Constructs a unary operation instruction.

Initializes the instruction with a basic block, a unique operation identifier, and the operation itself.

**Parameters**

| | |
|---|---|
| *bb_* | The basic block to which the instruction belongs. |
| *uniqueOp* | A unique identifier for the operation. |
| *op* | The actual unary operation (e.g., '!', '-', '∼'). |

### 5.25.3 Member Function Documentation

#### 5.25.3.1 gen_asm()

```
void IRInstrUnaryOp::gen_asm (
            ostream & o )  [virtual]
```

Generates the assembly code for this unary operation instruction.

This method generates the appropriate assembly code based on the specific unary operation (e.g., negation, logical NOT, bitwise complement).

**Parameters**

| | |
|---|---|
| *o* | The output stream where the generated assembly code will be written. |

Implements BaseIRInstr.

### 5.25.4 Member Data Documentation

#### 5.25.4.1 op

```
string IRInstrUnaryOp::op  [protected]
```

The actual unary operation (e.g., '!', '-', '∼').

#### 5.25.4.2 uniqueOp

```
string IRInstrUnaryOp::uniqueOp  [protected]
```

A unique identifier for the unary operation.

The documentation for this class was generated from the following files:

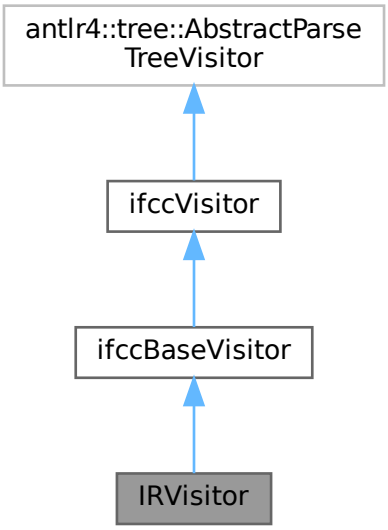
- IRInstrUnaryOp.h
- IRInstrUnaryOp.cpp

## 5.26 IRVisitor Class Reference

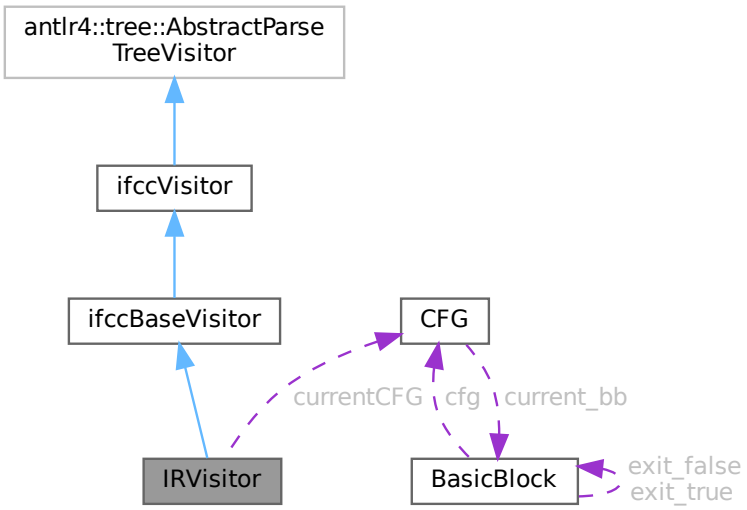
A visitor class for generating Intermediate Representation (IR) during parsing.

`#include <IRVisitor.h>`

Inheritance diagram for IRVisitor:




Collaboration diagram for IRVisitor:



### Public Member Functions

- IRVisitor (map< string, int > symbolsTable, int baseStackOffset)

  *Constructs an IRVisitor.*
- virtual antlrcpp::Any visitProg (ifccParser::ProgContext *ctx) override

  *Visits the program and starts the IR generation process.*
- virtual antlrcpp::Any visitReturn_stmt (ifccParser::Return_stmtContext *ctx) override

  *Visits a return statement and generates the IR.*
- virtual antlrcpp::Any visitAssign_stmt (ifccParser::Assign_stmtContext *ctx) override

  *Visits an assignment statement and generates the IR.*
- virtual antlrcpp::Any visitDecl_stmt (ifccParser::Decl_stmtContext *ctx) override

  *Visits a declaration statement and generates the IR.*
- antlrcpp::Any visitExpr (ifccParser::ExprContext *expr, bool isFirst)

  *Visits an expression and generates the IR.*
- virtual antlrcpp::Any visitAddsub (ifccParser::AddsubContext *ctx) override

  *Visits an addition or subtraction expression and generates the IR.*
- virtual antlrcpp::Any visitMuldiv (ifccParser::MuldivContext *ctx) override

  *Visits a multiplication or division expression and generates the IR.*
- virtual antlrcpp::Any visitBitwise (ifccParser::BitwiseContext *ctx) override

  *Visits a bitwise operation expression and generates the IR.*
- virtual antlrcpp::Any visitComp (ifccParser::CompContext *ctx) override

  *Visits a comparison expression and generates the IR.*
- virtual antlrcpp::Any visitUnary (ifccParser::UnaryContext *ctx) override

  *Visits a unary expression and generates the IR.*
- virtual antlrcpp::Any visitPre (ifccParser::PreContext *ctx) override

  *Visits a pre-unary operation (e.g., prefix increment/decrement) and generates the IR.*
- virtual antlrcpp::Any visitPost (ifccParser::PostContext *ctx) override

  *Visits a post-unary operation (e.g., postfix increment/decrement) and generates the IR.*
- void gen_asm (ostream &o)

  *Generates the assembly code for the IR.*
- void setCurrentCFG (CFG *currentCFG)

  *Sets the current control flow graph (CFG).*
- CFG * getCurrentCFG ()

  *Retrieves the current control flow graph (CFG).*
- map< string, CFG * > getCFGS ()

  *Retrieves the map of Control Flow Graphs (CFGs).*

## Public Member Functions inherited from ifccBaseVisitor

- virtual std::any visitAxiom (ifccParser::AxiomContext *ctx) override
- virtual std::any visitStatement (ifccParser::StatementContext *ctx) override
- virtual std::any visitIncrdecr_stmt (ifccParser::Incrdecr_stmtContext *ctx) override
- virtual std::any visitPar (ifccParser::ParContext *ctx) override
- virtual std::any visitConst (ifccParser::ConstContext *ctx) override
- virtual std::any visitVar (ifccParser::VarContext *ctx) override

### Protected Attributes

- map< string, CFG * > cfgs

  *A map of variable names to their corresponding Control Flow Graphs (CFGs).*
- CFG * currentCFG

  *The current control flow graph (CFG) being used.*

### 5.26.1 Detailed Description

A visitor class for generating Intermediate Representation (IR) during parsing.

This class extends the `ifccBaseVisitor` and is responsible for traversing the parsed code to generate the Intermediate Representation (IR), such as the Control Flow Graph (CFG), for the code being compiled.

### 5.26.2 Constructor & Destructor Documentation

#### 5.26.2.1 IRVisitor()

```
IRVisitor::IRVisitor (
            map< string, int > symbolsTable,
            int baseStackOffset )
```

Constructs an IRVisitor.

Initializes the IRVisitor with a symbols table and base stack offset.

**Parameters**

| | |
|---|---|
| *symbolsTable* | A map containing variable names and their associated stack offsets. |
| *baseStackOffset* | The base offset for the stack. |

### 5.26.3 Member Function Documentation

#### 5.26.3.1 gen_asm()

```
void IRVisitor::gen_asm (
            ostream & o )
```

Generates the assembly code for the IR.

This method generates assembly code from the Intermediate Representation (IR) for output.

**Parameters**

| | |
|---|---|
| *o* | The output stream where the assembly code will be written. |

#### 5.26.3.2 getCFGS()

```
map< string, CFG * > IRVisitor::getCFGS ( )
```

Retrieves the map of Control Flow Graphs (CFGs).

This function returns a map where the keys are string labels (e.g., function names or block labels) and the values are pointers to the corresponding `CFG` objects. These `CFG` objects represent the control flow graphs of different sections of the parsed program.

The returned map can be used for further analysis, visualization (e.g., by generating Graphviz `.dot` files), or manipulation of the program's control flow.

**Returns**

A `std::map` where the key is a string label representing the section of the program (such as a function name) and the value is a pointer to the corresponding `CFG` object.

#### 5.26.3.3 getCurrentCFG()

```
CFG * IRVisitor::getCurrentCFG ( )
```

Retrieves the current control flow graph (CFG).

This method retrieves the current CFG that is being used for the IR generation.

**Returns**

A pointer to the current CFG.

#### 5.26.3.4 setCurrentCFG()

```
void IRVisitor::setCurrentCFG (
            CFG * currentCFG )
```

Sets the current control flow graph (CFG).

This method sets the current CFG that is being used for the IR generation.

**Parameters**

| | |
|---|---|
| *currentCFG* | A pointer to the current CFG. |

#### 5.26.3.5 visitAddsub()

```
antlrcpp::Any IRVisitor::visitAddsub (
            ifccParser::AddsubContext * ctx )  [override], [virtual]
```

Visits an addition or subtraction expression and generates the IR.

This method processes addition and subtraction operations and generates the corresponding IR.

**Parameters**

| | |
|---|---|
| *ctx* | The context of the addition or subtraction expression. |

**Returns**

A result of the visit, typically unused.

Reimplemented from ifccBaseVisitor.

#### 5.26.3.6 visitAssign_stmt()

```
antlrcpp::Any IRVisitor::visitAssign_stmt (
            ifccParser::Assign_stmtContext * ctx )  [override], [virtual]
```

Visits an assignment statement and generates the IR.

This method processes assignment statements and generates the corresponding IR for the variable assignment.

**Parameters**

| | |
|---|---|
| *ctx* | The context of the assignment statement. |

**Returns**

A result of the visit, typically unused.

Reimplemented from ifccBaseVisitor.

#### 5.26.3.7 visitBitwise()

```
antlrcpp::Any IRVisitor::visitBitwise (
            ifccParser::BitwiseContext * ctx )  [override], [virtual]
```

Visits a bitwise operation expression and generates the IR.

This method processes bitwise operations like AND, OR, XOR, etc., and generates the corresponding IR.

**Parameters**

| | |
|---|---|
| *ctx* | The context of the bitwise operation expression. |

**Returns**

A result of the visit, typically unused.

Reimplemented from ifccBaseVisitor.

#### 5.26.3.8 visitComp()

```
antlrcpp::Any IRVisitor::visitComp (
            ifccParser::CompContext * ctx )  [override], [virtual]
```

Visits a comparison expression and generates the IR.

This method processes comparison operations (e.g., equality, greater-than) and generates the corresponding IR.

**Parameters**

| | |
|---|---|
| *ctx* | The context of the comparison expression. |

**Returns**

A result of the visit, typically unused.

Reimplemented from ifccBaseVisitor.

#### 5.26.3.9 visitDecl_stmt()

```
antlrcpp::Any IRVisitor::visitDecl_stmt (
            ifccParser::Decl_stmtContext * ctx )  [override], [virtual]
```

Visits a declaration statement and generates the IR.

This method processes declaration statements and generates the corresponding IR for the variable declaration.

**Parameters**

| | |
|---|---|
| *ctx* | The context of the declaration statement. |

**Returns**

A result of the visit, typically unused.

Reimplemented from ifccBaseVisitor.

#### 5.26.3.10 visitExpr()

```
antlrcpp::Any IRVisitor::visitExpr (
            ifccParser::ExprContext * expr,
            bool isFirst )
```

Visits an expression and generates the IR.

This method processes expressions and generates the corresponding IR for the expression.

**Parameters**

| | |
|---|---|
| *expr* | The expression context to generate IR for. |
| *isFirst* | A flag indicating whether this is the first expression in a sequence. |

**Returns**

A result of the visit, typically unused.

#### 5.26.3.11 visitMuldiv()

```
antlrcpp::Any IRVisitor::visitMuldiv (
            ifccParser::MuldivContext * ctx )  [override], [virtual]
```

Visits a multiplication or division expression and generates the IR.

This method processes multiplication and division operations and generates the corresponding IR.

**Parameters**

| | |
|---|---|
| *ctx* | The context of the multiplication or division expression. |

**Returns**

A result of the visit, typically unused.

Reimplemented from ifccBaseVisitor.

#### 5.26.3.12 visitPost()

```
antlrcpp::Any IRVisitor::visitPost (
            ifccParser::PostContext * ctx )  [override], [virtual]
```

Visits a post-unary operation (e.g., postfix increment/decrement) and generates the IR.

This method processes post-unary operations and generates the corresponding IR.

**Parameters**

| | |
|---|---|
| *ctx* | The context of the post-unary expression. |

**Returns**

A result of the visit, typically unused.

Reimplemented from ifccBaseVisitor.

#### 5.26.3.13 visitPre()

```
antlrcpp::Any IRVisitor::visitPre (
            ifccParser::PreContext * ctx )  [override], [virtual]
```

Visits a pre-unary operation (e.g., prefix increment/decrement) and generates the IR.

This method processes pre-unary operations and generates the corresponding IR.

**Parameters**

| | |
|---|---|
| *ctx* | The context of the pre-unary expression. |

**Returns**

A result of the visit, typically unused.

Reimplemented from ifccBaseVisitor.

#### 5.26.3.14 visitProg()

```
antlrcpp::Any IRVisitor::visitProg (
            ifccParser::ProgContext * ctx )  [override], [virtual]
```

Visits the program and starts the IR generation process.

This method starts the process of visiting the program node, generating the IR for the entire program.

**Parameters**

| | |
|---|---|
| *ctx* | The context of the program. |

**Returns**

A result of the visit, typically unused.

Reimplemented from ifccBaseVisitor.

#### 5.26.3.15 visitReturn_stmt()

```
antlrcpp::Any IRVisitor::visitReturn_stmt (
            ifccParser::Return_stmtContext * ctx )  [override], [virtual]
```

Visits a return statement and generates the IR.

This method processes return statements and generates the corresponding IR for the return operation.

**Parameters**

| | |
|---|---|
| *ctx* | The context of the return statement. |

**Returns**

A result of the visit, typically unused.

Reimplemented from ifccBaseVisitor.

#### 5.26.3.16 visitUnary()

```
antlrcpp::Any IRVisitor::visitUnary (
            ifccParser::UnaryContext * ctx )  [override], [virtual]
```

Visits a unary expression and generates the IR.

This method processes unary operations (e.g., negation or logical NOT) and generates the corresponding IR.

**Parameters**

| | |
|---|---|
| *ctx* | The context of the unary expression. |

**Returns**

A result of the visit, typically unused.

Reimplemented from ifccBaseVisitor.

### 5.26.4 Member Data Documentation

#### 5.26.4.1 cfgs

```
map<string, CFG *> IRVisitor::cfgs  [protected]
```

A map of variable names to their corresponding Control Flow Graphs (CFGs).

#### 5.26.4.2 currentCFG

```
CFG* IRVisitor::currentCFG  [protected]
```

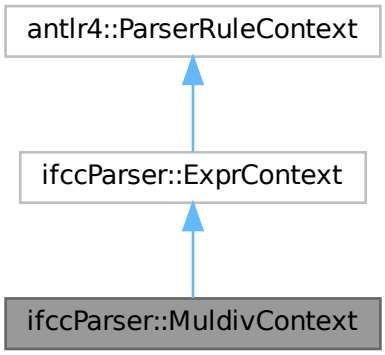
The current control flow graph (CFG) being used.

The documentation for this class was generated from the following files:

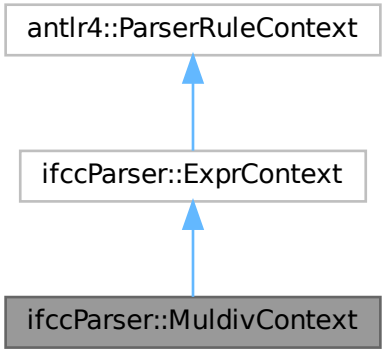
- IRVisitor.h
- IRVisitor.cpp

## 5.27 ifccParser::MuldivContext Class Reference

Inheritance diagram for ifccParser::MuldivContext:




Collaboration diagram for ifccParser::MuldivContext:




### Public Member Functions

- **MuldivContext** (ExprContext ∗ctx)
- std::vector< ExprContext ∗ > **expr** ()
- ExprContext ∗ **expr** (size_t i)
- virtual std::any **accept** (antlr4::tree::ParseTreeVisitor ∗visitor) override

### Public Member Functions inherited from ifccParser::ExprContext

- **ExprContext** (antlr4::ParserRuleContext ∗parent, size_t invokingState)
- void **copyFrom** (ExprContext ∗context)
- virtual size_t **getRuleIndex** () const override

**Public Attributes**

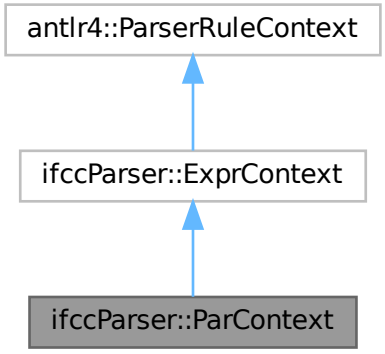
- antlr4::Token * **OP** = nullptr

The documentation for this class was generated from the following files:

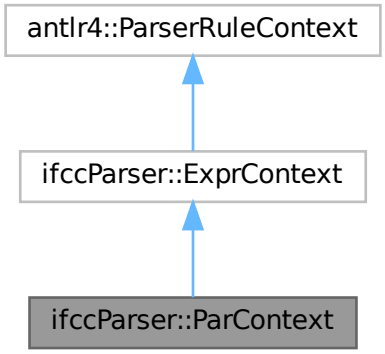
- ifccParser.h
- ifccParser.cpp

## 5.28 ifccParser::ParContext Class Reference

Inheritance diagram for ifccParser::ParContext:




Collaboration diagram for ifccParser::ParContext:



#### Public Member Functions

- **ParContext** (ExprContext ∗ctx)
- ExprContext ∗ **expr** ()
- virtual std::any **accept** (antlr4::tree::ParseTreeVisitor ∗visitor) override

### Public Member Functions inherited from ifccParser::ExprContext

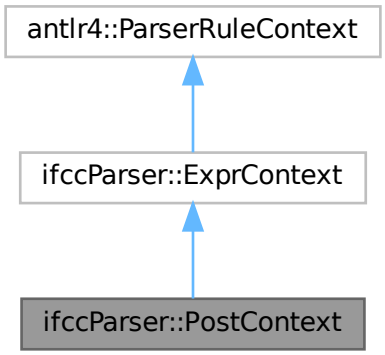
- **ExprContext** (antlr4::ParserRuleContext ∗parent, size_t invokingState)
- void **copyFrom** (ExprContext ∗context)
- virtual size_t **getRuleIndex** () const override

The documentation for this class was generated from the following files:

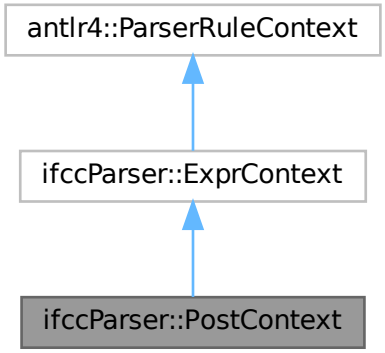
- ifccParser.h
- ifccParser.cpp

## 5.29 ifccParser::PostContext Class Reference

Inheritance diagram for ifccParser::PostContext:



Collaboration diagram for ifccParser::PostContext:




### Public Member Functions

- **PostContext** (ExprContext ∗ctx)
- antlr4::tree::TerminalNode ∗ **VAR** ()
- virtual std::any **accept** (antlr4::tree::ParseTreeVisitor ∗visitor) override

## Public Member Functions inherited from ifccParser::ExprContext

- **ExprContext** (antlr4::ParserRuleContext ∗parent, size_t invokingState)
- void **copyFrom** (ExprContext ∗context)
- virtual size_t **getRuleIndex** () const override

### Public Attributes

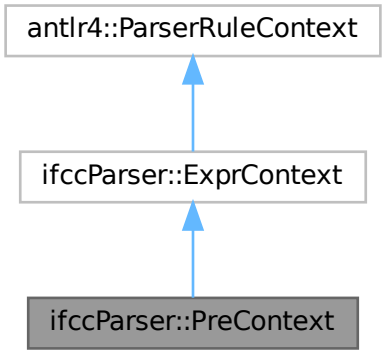
- antlr4::Token ∗ **OP** = nullptr

The documentation for this class was generated from the following files:

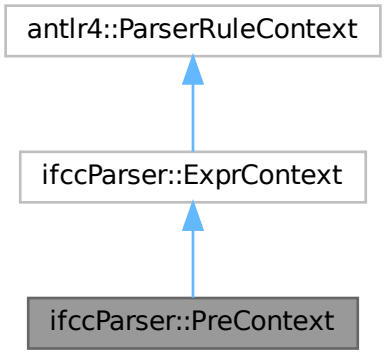
- ifccParser.h
- ifccParser.cpp

## 5.30 ifccParser::PreContext Class Reference

Inheritance diagram for ifccParser::PreContext:




Collaboration diagram for ifccParser::PreContext:




### Public Member Functions

- **PreContext** (ExprContext ∗ctx)
- antlr4::tree::TerminalNode ∗ **VAR** ()
- virtual std::any **accept** (antlr4::tree::ParseTreeVisitor ∗visitor) override

### Public Member Functions inherited from ifccParser::ExprContext

- **ExprContext** (antlr4::ParserRuleContext ∗parent, size_t invokingState)
- void **copyFrom** (ExprContext ∗context)
- virtual size_t **getRuleIndex** () const override

### Public Attributes

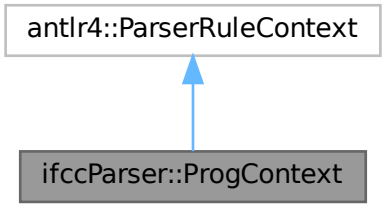
- antlr4::Token * **OP** = nullptr

The documentation for this class was generated from the following files:

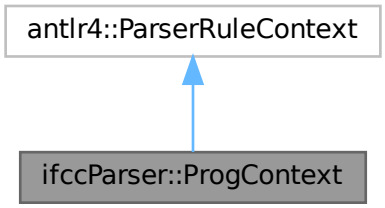
- ifccParser.h
- ifccParser.cpp

## 5.31 ifccParser::ProgContext Class Reference

Inheritance diagram for ifccParser::ProgContext:




Collaboration diagram for ifccParser::ProgContext:




### Public Member Functions

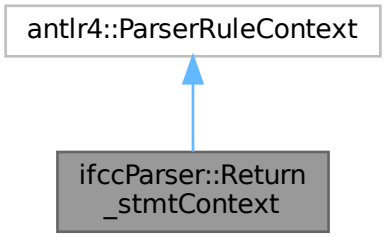
- **ProgContext** (antlr4::ParserRuleContext *parent, size_t invokingState)
- virtual size_t **getRuleIndex** () const override
- antlr4::tree::TerminalNode * **TYPE** ()
- Return_stmtContext * **return_stmt** ()
- std::vector< StatementContext * > **statement** ()
- StatementContext * **statement** (size_t i)
- virtual std::any **accept** (antlr4::tree::ParseTreeVisitor *visitor) override

The documentation for this class was generated from the following files:

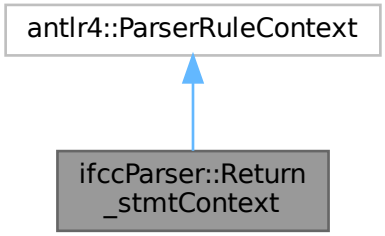
- ifccParser.h
- ifccParser.cpp

## 5.32 ifccParser::Return_stmtContext Class Reference

Inheritance diagram for ifccParser::Return_stmtContext:




Collaboration diagram for ifccParser::Return_stmtContext:




### Public Member Functions

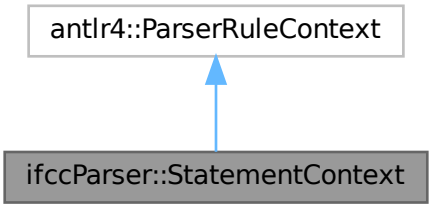
- **Return_stmtContext** (antlr4::ParserRuleContext ∗parent, size_t invokingState)
- virtual size_t **getRuleIndex** () const override
- antlr4::tree::TerminalNode ∗ **RETURN** ()
- ExprContext ∗ **expr** ()
- Assign_stmtContext ∗ **assign_stmt** ()
- virtual std::any **accept** (antlr4::tree::ParseTreeVisitor ∗visitor) override

The documentation for this class was generated from the following files:

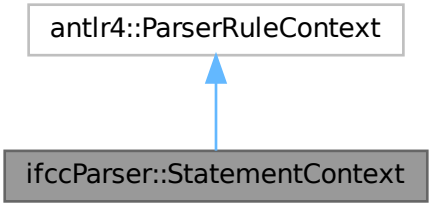
- ifccParser.h
- ifccParser.cpp

## 5.33 ifccParser::StatementContext Class Reference

Inheritance diagram for ifccParser::StatementContext:




Collaboration diagram for ifccParser::StatementContext:




### Public Member Functions

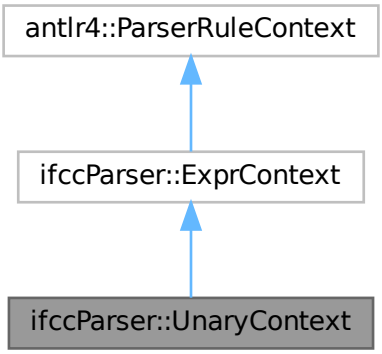
- **StatementContext** (antlr4::ParserRuleContext ∗parent, size_t invokingState)
- virtual size_t **getRuleIndex** () const override
- Decl_stmtContext ∗ **decl_stmt** ()
- Assign_stmtContext ∗ **assign_stmt** ()
- Incrdecr_stmtContext ∗ **incrdecr_stmt** ()
- virtual std::any **accept** (antlr4::tree::ParseTreeVisitor ∗visitor) override

The documentation for this class was generated from the following files:

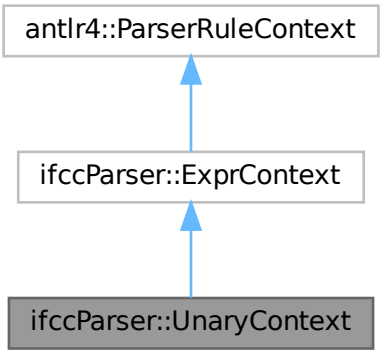
- ifccParser.h
- ifccParser.cpp

## 5.34 ifccParser::UnaryContext Class Reference

Inheritance diagram for ifccParser::UnaryContext:




Collaboration diagram for ifccParser::UnaryContext:




### Public Member Functions

- **UnaryContext** (ExprContext ∗ctx)
- ExprContext ∗ **expr** ()
- virtual std::any **accept** (antlr4::tree::ParseTreeVisitor ∗visitor) override

### Public Member Functions inherited from ifccParser::ExprContext

- **ExprContext** (antlr4::ParserRuleContext ∗parent, size_t invokingState)
- void **copyFrom** (ExprContext ∗context)
- virtual size_t **getRuleIndex** () const override

**Public Attributes**

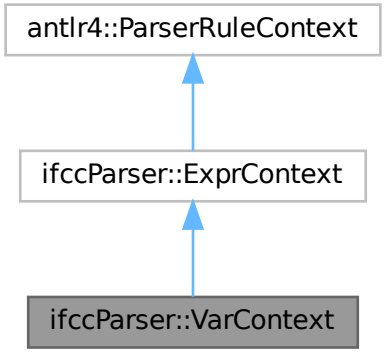
- antlr4::Token * **OP** = nullptr

The documentation for this class was generated from the following files:

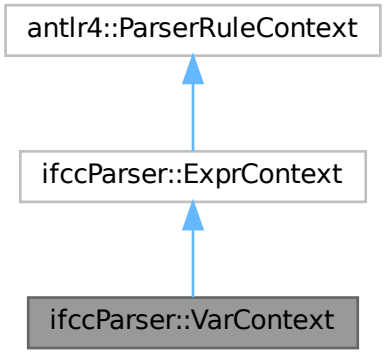
- ifccParser.h
- ifccParser.cpp

## 5.35 ifccParser::VarContext Class Reference

Inheritance diagram for ifccParser::VarContext:




Collaboration diagram for ifccParser::VarContext:



### Public Member Functions

- **VarContext** (ExprContext ∗ctx)
- antlr4::tree::TerminalNode ∗ **VAR** ()
- virtual std::any **accept** (antlr4::tree::ParseTreeVisitor ∗visitor) override

## Public Member Functions inherited from ifccParser::ExprContext

- **ExprContext** (antlr4::ParserRuleContext ∗parent, size_t invokingState)
- void **copyFrom** (ExprContext ∗context)
- virtual size_t **getRuleIndex** () const override

The documentation for this class was generated from the following files:

- ifccParser.h
- ifccParser.cpp

# Chapter 6

# File Documentation

## 6.1 CodeCheckVisitor.h

```
00001 #pragma once
00002
00003 #include "antlr4-runtime.h"
00004 #include "generated/ifccBaseVisitor.h"
00005 #include <map>
00006
00007 using namespace std;
00008
00016 class CodeCheckVisitor : public ifccBaseVisitor
00017 {
00018 public:
00028         virtual antlrcpp::Any visitReturn_stmt(ifccParser::Return_stmtContext *ctx) override;
00029
00039         virtual antlrcpp::Any visitAssign_stmt(ifccParser::Assign_stmtContext *ctx) override;
00040
00050         virtual antlrcpp::Any visitDecl_stmt(ifccParser::Decl_stmtContext *ctx) override;
00051
00061         antlrcpp::Any visitExpr(ifccParser::ExprContext *expr);
00062
00072         virtual antlrcpp::Any visitAddsub(ifccParser::AddsubContext *ctx) override;
00073
00083         virtual antlrcpp::Any visitMuldiv(ifccParser::MuldivContext *ctx) override;
00084
00094         virtual antlrcpp::Any visitBitwise(ifccParser::BitwiseContext *ctx) override;
00095
00106         virtual antlrcpp::Any visitComp(ifccParser::CompContext *ctx) override;
00107
00117         virtual antlrcpp::Any visitUnary(ifccParser::UnaryContext *ctx) override;
00118
00127         virtual antlrcpp::Any visitPre(ifccParser::PreContext *ctx) override;
00128
00137         virtual antlrcpp::Any visitPost(ifccParser::PostContext *ctx) override;
00138
00144         map<string, int> getSymbolsTable() const
00145         {
00146                 return symbolsTable;
00147         }
00148
00154         map<string, bool> getIsUsed() const { return isUsed; }
00155
00161         int getCurrentOffset() const { return currentOffset; }
00162
00163 protected:
00165         map<string, int> symbolsTable;
00166
00168         map<string, bool> isUsed;
00169
00171         map<string, bool> hasAValue;
00172
00174         int currentOffset = 0;
00175 };
```

## 6.2 ifccBaseVisitor.h

```
00001
```

```
00002 // Generated from ifcc.g4 by ANTLR 4.13.2
00003
00004 #pragma once
00005
00006
00007 #include "antlr4-runtime.h"
00008 #include "ifccVisitor.h"
00009
00010
00015 class  ifccBaseVisitor : public ifccVisitor {
00016 public:
00017
00018   virtual std::any visitAxiom(ifccParser::AxiomContext *ctx) override {
00019     return visitChildren(ctx);
00020   }
00021
00022   virtual std::any visitProg(ifccParser::ProgContext *ctx) override {
00023     return visitChildren(ctx);
00024   }
00025
00026   virtual std::any visitStatement(ifccParser::StatementContext *ctx) override {
00027     return visitChildren(ctx);
00028   }
00029
00030   virtual std::any visitDecl_stmt(ifccParser::Decl_stmtContext *ctx) override {
00031     return visitChildren(ctx);
00032   }
00033
00034   virtual std::any visitAssign_stmt(ifccParser::Assign_stmtContext *ctx) override {
00035     return visitChildren(ctx);
00036   }
00037
00038   virtual std::any visitIncrdecr_stmt(ifccParser::Incrdecr_stmtContext *ctx) override {
00039     return visitChildren(ctx);
00040   }
00041
00042   virtual std::any visitReturn_stmt(ifccParser::Return_stmtContext *ctx) override {
00043     return visitChildren(ctx);
00044   }
00045
00046   virtual std::any visitPar(ifccParser::ParContext *ctx) override {
00047     return visitChildren(ctx);
00048   }
00049
00050   virtual std::any visitComp(ifccParser::CompContext *ctx) override {
00051     return visitChildren(ctx);
00052   }
00053
00054   virtual std::any visitPre(ifccParser::PreContext *ctx) override {
00055     return visitChildren(ctx);
00056   }
00057
00058   virtual std::any visitConst(ifccParser::ConstContext *ctx) override {
00059     return visitChildren(ctx);
00060   }
00061
00062   virtual std::any visitPost(ifccParser::PostContext *ctx) override {
00063     return visitChildren(ctx);
00064   }
00065
00066   virtual std::any visitVar(ifccParser::VarContext *ctx) override {
00067     return visitChildren(ctx);
00068   }
00069
00070   virtual std::any visitBitwise(ifccParser::BitwiseContext *ctx) override {
00071     return visitChildren(ctx);
00072   }
00073
00074   virtual std::any visitAddsub(ifccParser::AddsubContext *ctx) override {
00075     return visitChildren(ctx);
00076   }
00077
00078   virtual std::any visitUnary(ifccParser::UnaryContext *ctx) override {
00079     return visitChildren(ctx);
00080   }
00081
00082   virtual std::any visitMuldiv(ifccParser::MuldivContext *ctx) override {
00083     return visitChildren(ctx);
00084   }
00085
00086
00087 };
00088
```

## 6.3 ifccLexer.h

```
00001
00002 // Generated from ifcc.g4 by ANTLR 4.13.2
00003
00004 #pragma once
00005
00006
00007 #include "antlr4-runtime.h"
00008
00009
00010
00011
00012 class  ifccLexer : public antlr4::Lexer {
00013 public:
00014   enum {
00015     T__0 = 1, T__1 = 2, T__2 = 3, T__3 = 4, T__4 = 5, T__5 = 6, T__6 = 7,
00016     T__7 = 8, T__8 = 9, T__9 = 10, T__10 = 11, T__11 = 12, T__12 = 13, T__13 = 14,
00017     T__14 = 15, T__15 = 16, T__16 = 17, T__17 = 18, T__18 = 19, T__19 = 20,
00018     T__20 = 21, T__21 = 22, T__22 = 23, T__23 = 24, T__24 = 25, T__25 = 26,
00019     OPU = 27, RETURN = 28, TYPE = 29, VAR = 30, CONST = 31, COMMENT = 32,
00020     DIRECTIVE = 33, WS = 34
00021   };
00022
00023   explicit ifccLexer(antlr4::CharStream *input);
00024
00025   ~ifccLexer() override;
00026
00027
00028   std::string getGrammarFileName() const override;
00029
00030   const std::vector<std::string>& getRuleNames() const override;
00031
00032   const std::vector<std::string>& getChannelNames() const override;
00033
00034   const std::vector<std::string>& getModeNames() const override;
00035
00036   const antlr4::dfa::Vocabulary& getVocabulary() const override;
00037
00038   antlr4::atn::SerializedATNView getSerializedATN() const override;
00039
00040   const antlr4::atn::ATN& getATN() const override;
00041
00042   // By default the static state used to implement the lexer is lazily initialized during the first
00043   // call to the constructor. You can call this function if you wish to initialize the static state
00044   // ahead of time.
00045   static void initialize();
00046
00047 private:
00048
00049   // Individual action functions triggered by action() above.
00050
00051   // Individual semantic predicate functions triggered by sempred() above.
00052
00053 };
00054
```

## 6.4 ifccParser.h

```
00001
00002 // Generated from ifcc.g4 by ANTLR 4.13.2
00003
00004 #pragma once
00005
00006
00007 #include "antlr4-runtime.h"
00008
00009
00010
00011
00012 class  ifccParser : public antlr4::Parser {
00013 public:
00014   enum {
00015     T__0 = 1, T__1 = 2, T__2 = 3, T__3 = 4, T__4 = 5, T__5 = 6, T__6 = 7,
00016     T__7 = 8, T__8 = 9, T__9 = 10, T__10 = 11, T__11 = 12, T__12 = 13, T__13 = 14,
00017     T__14 = 15, T__15 = 16, T__16 = 17, T__17 = 18, T__18 = 19, T__19 = 20,
00018     T__20 = 21, T__21 = 22, T__22 = 23, T__23 = 24, T__24 = 25, T__25 = 26,
00019     OPU = 27, RETURN = 28, TYPE = 29, VAR = 30, CONST = 31, COMMENT = 32,
00020     DIRECTIVE = 33, WS = 34
00021   };
00022
00023   enum {
00024     RuleAxiom = 0, RuleProg = 1, RuleStatement = 2, RuleDecl_stmt = 3, RuleAssign_stmt = 4,
```

```
00025     RuleIncrdecr_stmt = 5, RuleReturn_stmt = 6, RuleExpr = 7
00026   };
00027
00028   explicit ifccParser(antlr4::TokenStream *input);
00029
00030   ifccParser(antlr4::TokenStream *input, const antlr4::atn::ParserATNSimulatorOptions &options);
00031
00032   ~ifccParser() override;
00033
00034   std::string getGrammarFileName() const override;
00035
00036   const antlr4::atn::ATN& getATN() const override;
00037
00038   const std::vector<std::string>& getRuleNames() const override;
00039
00040   const antlr4::dfa::Vocabulary& getVocabulary() const override;
00041
00042   antlr4::atn::SerializedATNView getSerializedATN() const override;
00043
00044
00045   class AxiomContext;
00046   class ProgContext;
00047   class StatementContext;
00048   class Decl_stmtContext;
00049   class Assign_stmtContext;
00050   class Incrdecr_stmtContext;
00051   class Return_stmtContext;
00052   class ExprContext;
00053
00054   class  AxiomContext : public antlr4::ParserRuleContext {
00055   public:
00056     AxiomContext(antlr4::ParserRuleContext *parent, size_t invokingState);
00057     virtual size_t getRuleIndex() const override;
00058     ProgContext *prog();
00059     antlr4::tree::TerminalNode *EOF();
00060
00061
00062     virtual std::any accept(antlr4::tree::ParseTreeVisitor *visitor) override;
00063
00064   };
00065
00066   AxiomContext* axiom();
00067
00068   class  ProgContext : public antlr4::ParserRuleContext {
00069   public:
00070     ProgContext(antlr4::ParserRuleContext *parent, size_t invokingState);
00071     virtual size_t getRuleIndex() const override;
00072     antlr4::tree::TerminalNode *TYPE();
00073     Return_stmtContext *return_stmt();
00074     std::vector<StatementContext *> statement();
00075     StatementContext* statement(size_t i);
00076
00077
00078     virtual std::any accept(antlr4::tree::ParseTreeVisitor *visitor) override;
00079
00080   };
00081
00082   ProgContext* prog();
00083
00084   class  StatementContext : public antlr4::ParserRuleContext {
00085   public:
00086     StatementContext(antlr4::ParserRuleContext *parent, size_t invokingState);
00087     virtual size_t getRuleIndex() const override;
00088     Decl_stmtContext *decl_stmt();
00089     Assign_stmtContext *assign_stmt();
00090     Incrdecr_stmtContext *incrdecr_stmt();
00091
00092
00093     virtual std::any accept(antlr4::tree::ParseTreeVisitor *visitor) override;
00094
00095   };
00096
00097   StatementContext* statement();
00098
00099   class  Decl_stmtContext : public antlr4::ParserRuleContext {
00100   public:
00101     Decl_stmtContext(antlr4::ParserRuleContext *parent, size_t invokingState);
00102     virtual size_t getRuleIndex() const override;
00103     antlr4::tree::TerminalNode *TYPE();
00104     std::vector<antlr4::tree::TerminalNode *> VAR();
00105     antlr4::tree::TerminalNode* VAR(size_t i);
00106     std::vector<ExprContext *> expr();
00107     ExprContext* expr(size_t i);
00108
00109
00110     virtual std::any accept(antlr4::tree::ParseTreeVisitor *visitor) override;
00111
```

```
00112   };
00113
00114   Decl_stmtContext* decl_stmt();
00115
00116   class  Assign_stmtContext : public antlr4::ParserRuleContext {
00117   public:
00118     Assign_stmtContext(antlr4::ParserRuleContext *parent, size_t invokingState);
00119     virtual size_t getRuleIndex() const override;
00120     antlr4::tree::TerminalNode *VAR();
00121     ExprContext *expr();
00122
00123
00124     virtual std::any accept(antlr4::tree::ParseTreeVisitor *visitor) override;
00125
00126   };
00127
00128   Assign_stmtContext* assign_stmt();
00129
00130   class  Incrdecr_stmtContext : public antlr4::ParserRuleContext {
00131   public:
00132     antlr4::Token *OP = nullptr;
00133     Incrdecr_stmtContext(antlr4::ParserRuleContext *parent, size_t invokingState);
00134     virtual size_t getRuleIndex() const override;
00135     antlr4::tree::TerminalNode *VAR();
00136
00137
00138     virtual std::any accept(antlr4::tree::ParseTreeVisitor *visitor) override;
00139
00140   };
00141
00142   Incrdecr_stmtContext* incrdecr_stmt();
00143
00144   class  Return_stmtContext : public antlr4::ParserRuleContext {
00145   public:
00146     Return_stmtContext(antlr4::ParserRuleContext *parent, size_t invokingState);
00147     virtual size_t getRuleIndex() const override;
00148     antlr4::tree::TerminalNode *RETURN();
00149     ExprContext *expr();
00150     Assign_stmtContext *assign_stmt();
00151
00152
00153     virtual std::any accept(antlr4::tree::ParseTreeVisitor *visitor) override;
00154
00155   };
00156
00157   Return_stmtContext* return_stmt();
00158
00159   class  ExprContext : public antlr4::ParserRuleContext {
00160   public:
00161     ExprContext(antlr4::ParserRuleContext *parent, size_t invokingState);
00162
00163     ExprContext() = default;
00164     void copyFrom(ExprContext *context);
00165     using antlr4::ParserRuleContext::copyFrom;
00166
00167     virtual size_t getRuleIndex() const override;
00168
00169
00170   };
00171
00172   class  ParContext : public ExprContext {
00173   public:
00174     ParContext(ExprContext *ctx);
00175
00176     ExprContext *expr();
00177
00178     virtual std::any accept(antlr4::tree::ParseTreeVisitor *visitor) override;
00179   };
00180
00181   class  CompContext : public ExprContext {
00182   public:
00183     CompContext(ExprContext *ctx);
00184
00185     antlr4::Token *OP = nullptr;
00186     std::vector<ExprContext *> expr();
00187     ExprContext* expr(size_t i);
00188
00189     virtual std::any accept(antlr4::tree::ParseTreeVisitor *visitor) override;
00190   };
00191
00192   class  PreContext : public ExprContext {
00193   public:
00194     PreContext(ExprContext *ctx);
00195
00196     antlr4::Token *OP = nullptr;
00197     antlr4::tree::TerminalNode *VAR();
00198
```

```
00199     virtual std::any accept(antlr4::tree::ParseTreeVisitor *visitor) override;
00200   };
00201
00202   class  ConstContext : public ExprContext {
00203   public:
00204     ConstContext(ExprContext *ctx);
00205
00206     antlr4::tree::TerminalNode *CONST();
00207
00208     virtual std::any accept(antlr4::tree::ParseTreeVisitor *visitor) override;
00209   };
00210
00211   class  PostContext : public ExprContext {
00212   public:
00213     PostContext(ExprContext *ctx);
00214
00215     antlr4::Token *OP = nullptr;
00216     antlr4::tree::TerminalNode *VAR();
00217
00218     virtual std::any accept(antlr4::tree::ParseTreeVisitor *visitor) override;
00219   };
00220
00221   class  VarContext : public ExprContext {
00222   public:
00223     VarContext(ExprContext *ctx);
00224
00225     antlr4::tree::TerminalNode *VAR();
00226
00227     virtual std::any accept(antlr4::tree::ParseTreeVisitor *visitor) override;
00228   };
00229
00230   class  BitwiseContext : public ExprContext {
00231   public:
00232     BitwiseContext(ExprContext *ctx);
00233
00234     antlr4::Token *OP = nullptr;
00235     std::vector<ExprContext *> expr();
00236     ExprContext* expr(size_t i);
00237
00238     virtual std::any accept(antlr4::tree::ParseTreeVisitor *visitor) override;
00239   };
00240
00241   class  AddsubContext : public ExprContext {
00242   public:
00243     AddsubContext(ExprContext *ctx);
00244
00245     antlr4::Token *OP = nullptr;
00246     std::vector<ExprContext *> expr();
00247     ExprContext* expr(size_t i);
00248
00249     virtual std::any accept(antlr4::tree::ParseTreeVisitor *visitor) override;
00250   };
00251
00252   class  UnaryContext : public ExprContext {
00253   public:
00254     UnaryContext(ExprContext *ctx);
00255
00256     antlr4::Token *OP = nullptr;
00257     ExprContext *expr();
00258
00259     virtual std::any accept(antlr4::tree::ParseTreeVisitor *visitor) override;
00260   };
00261
00262   class  MuldivContext : public ExprContext {
00263   public:
00264     MuldivContext(ExprContext *ctx);
00265
00266     antlr4::Token *OP = nullptr;
00267     std::vector<ExprContext *> expr();
00268     ExprContext* expr(size_t i);
00269
00270     virtual std::any accept(antlr4::tree::ParseTreeVisitor *visitor) override;
00271   };
00272
00273   ExprContext* expr();
00274   ExprContext* expr(int precedence);
00275
00276   bool sempred(antlr4::RuleContext *_localctx, size_t ruleIndex, size_t predicateIndex) override;
00277
00278   bool exprSempred(ExprContext *_localctx, size_t predicateIndex);
00279
00280   // By default the static state used to implement the parser is lazily initialized during the first
00281   // call to the constructor. You can call this function if you wish to initialize the static state
00282   // ahead of time.
00283   static void initialize();
00284
00285 private:
```

```
00286 };
00287
```

## 6.5 ifccVisitor.h

```
00001
00002 // Generated from ifcc.g4 by ANTLR 4.13.2
00003
00004 #pragma once
00005
00006
00007 #include "antlr4-runtime.h"
00008 #include "ifccParser.h"
00009
00010
00011
00016 class  ifccVisitor : public antlr4::tree::AbstractParseTreeVisitor {
00017 public:
00018
00022     virtual std::any visitAxiom(ifccParser::AxiomContext *context) = 0;
00023
00024     virtual std::any visitProg(ifccParser::ProgContext *context) = 0;
00025
00026     virtual std::any visitStatement(ifccParser::StatementContext *context) = 0;
00027
00028     virtual std::any visitDecl_stmt(ifccParser::Decl_stmtContext *context) = 0;
00029
00030     virtual std::any visitAssign_stmt(ifccParser::Assign_stmtContext *context) = 0;
00031
00032     virtual std::any visitIncrdecr_stmt(ifccParser::Incrdecr_stmtContext *context) = 0;
00033
00034     virtual std::any visitReturn_stmt(ifccParser::Return_stmtContext *context) = 0;
00035
00036     virtual std::any visitPar(ifccParser::ParContext *context) = 0;
00037
00038     virtual std::any visitComp(ifccParser::CompContext *context) = 0;
00039
00040     virtual std::any visitPre(ifccParser::PreContext *context) = 0;
00041
00042     virtual std::any visitConst(ifccParser::ConstContext *context) = 0;
00043
00044     virtual std::any visitPost(ifccParser::PostContext *context) = 0;
00045
00046     virtual std::any visitVar(ifccParser::VarContext *context) = 0;
00047
00048     virtual std::any visitBitwise(ifccParser::BitwiseContext *context) = 0;
00049
00050     virtual std::any visitAddsub(ifccParser::AddsubContext *context) = 0;
00051
00052     virtual std::any visitUnary(ifccParser::UnaryContext *context) = 0;
00053
00054     virtual std::any visitMuldiv(ifccParser::MuldivContext *context) = 0;
00055
00056
00057 };
00058
```

## 6.6 BasicBlock.h

```
00001 #pragma once
00002
00003 #include "CFG.h"
00004 #include "Instr/BaseIRInstr.h"
00005 #include <vector>
00006
00007 using namespace std;
00008
00009 class CFG;
00010 class BaseIRInstr;
00011
00018 class BasicBlock
00019 {
00020 public:
00029   BasicBlock(CFG *cfg, string entry_label);
00030
00038   void gen_asm(ostream &o);
00039
00047   void add_IRInstr(BaseIRInstr *instr);
00048
```

```
00056   CFG *getCFG();
00057
00066   string getLabel();
00067
00077   vector<BaseIRInstr *> getInstr();
00078
00088   void setExitTrue(BasicBlock *bb);
00089
00099   void setExitFalse(BasicBlock *bb);
00100
00109   BasicBlock *getExitTrue();
00110
00119   BasicBlock *getExitFalse();
00120
00121 protected:
00125   BasicBlock *exit_true;
00126
00130   BasicBlock *exit_false;
00131
00133   string label;
00134
00136   CFG *cfg;
00137
00139   vector<BaseIRInstr *> instrs;
00140 };
```

## 6.7 CFG.h

```
00001 #pragma once
00002 #include "BasicBlock.h"
00003 #include <ostream>
00004 #include <string>
00005 #include <map>
00006 #include <vector>
00007
00008 using namespace std;
00009
00010 class BasicBlock;
00011
00019 class CFG
00020 {
00021 public:
00032     CFG(string label, map<string, int> SymbolIndex, int initialNextFreeSymbolIndex);
00033
00041     void add_bb(BasicBlock *bb);
00042
00050     void gen_asm(ostream &o);
00051
00060     void gen_asm_prologue(ostream &o);
00061
00070     void gen_asm_epilogue(ostream &o);
00071
00079     string create_new_tempvar();
00080
00089     int get_var_index(string name);
00090
00098     BasicBlock *getCurrentBasicBlock();
00099
00107     void setCurrentBasicBlock(BasicBlock *bb);
00108
00114     void resetNextFreeSymbolIndex();
00115
00125     void gen_cfg_graphviz(ostream &o);
00126
00136     string getLabel();
00137
00138 protected:
00140     map<string, int> SymbolIndex;
00141
00143     int nextFreeSymbolIndex;
00144
00146     const int initialTempPos;
00147
00149     vector<BasicBlock *> bbs;
00150
00152     BasicBlock *current_bb;
00153
00155     string label;
00156 };
```

## 6.8 BaseIRInstr.h

```
00001 #pragma once
00002
00003 #include "../BasicBlock.h"
00004 #include <string>
00005 #include <ostream>
00006
00007 using namespace std;
00008
00009 class BasicBlock;
00010
00017 class BaseIRInstr
00018 {
00019 public:
00027     BaseIRInstr(BasicBlock *bb_) : bb(bb_) {}
00028
00034     BasicBlock *getBB();
00035
00044     virtual void gen_asm(ostream &o) = 0;
00045
00046 protected:
00048     BasicBlock *bb;
00049 };
```

## 6.9 IRInstrArithmeticOp.h

```
00001 #pragma once
00002
00003 #include "BaseIRInstr.h"
00004 #include "IRInstrBinaryOp.h"
00005
00013 class IRInstrArithmeticOp : public IRInstrBinaryOp
00014 {
00015 public:
00026     IRInstrArithmeticOp(BasicBlock *bb_, string firstOp, string secondOp, string op)
00027         : IRInstrBinaryOp(bb_, firstOp, secondOp, op) {}
00028
00037     virtual void gen_asm(ostream &o);
00038
00039 private:
00047     void handleAddition(ostream &o);
00048
00056     void handleSubstraction(ostream &o);
00057
00065     void handleMult(ostream &o);
00066
00074     void handleDiv(ostream &o);
00075
00083     void handleModulo(ostream &o);
00084
00092     void handleBitwiseAnd(ostream &o);
00093
00101     void handleBitwiseOr(ostream &o);
00102
00110     void handleBitwiseXor(ostream &o);
00111 };
```

## 6.10 IRInstrBinaryOp.h

```
00001 #pragma once
00002
00003 #include "BaseIRInstr.h"
00004
00012 class IRInstrBinaryOp : public BaseIRInstr
00013 {
00014 public:
00025     IRInstrBinaryOp(BasicBlock *bb_, string firstOp, string secondOp, string op)
00026         : BaseIRInstr(bb_), firstOp(firstOp), secondOp(secondOp), op(op) {}
00027
00036     virtual void gen_asm(ostream &o) = 0;
00037
00038 protected:
00040     string firstOp;
00041
00043     string secondOp;
00044
00046     string op;
00047 };
```

## 6.11 IRInstrClean.h

```
00001 #pragma once
00002
00003 #include "BaseIRInstr.h"
00004 #include <ostream>
00005
00015 class IRInstrClean : public BaseIRInstr
00016 {
00017 public:
00026     IRInstrClean(BasicBlock *bb_) : BaseIRInstr(bb_) {};
00027
00039     virtual void gen_asm(std::ostream &o) override;
00040 };
```

## 6.12 IRInstrComp.h

```
00001 #pragma once
00002
00003 #include "IRInstrBinaryOp.h"
00004 #include <ostream>
00005
00012 class IRInstrComp : public IRInstrBinaryOp
00013 {
00014 public:
00025     IRInstrComp(BasicBlock *bb_, string firstOp, string secondOp, string op)
00026         : IRInstrBinaryOp(bb_, firstOp, secondOp, op) {}
00027
00036     virtual void gen_asm(ostream &o) override;
00037
00038 private:
00046     void handleCompEq(ostream &o);
00047
00055     void handleCompNotEq(ostream &o);
00056
00064     void handleGreater(ostream &o);
00065
00073     void handleGreaterOrEqual(ostream &o);
00074
00082     void handleLower(ostream &o);
00083
00091     void handleLowerOrEqual(ostream &o);
00092 };
```

## 6.13 IRInstrLoadConst.h

```
00001 #pragma once
00002
00003 #include "BaseIRInstr.h"
00004 #include <string>
00005
00012 class IRInstrLoadConst : public BaseIRInstr
00013 {
00014 public:
00025     IRInstrLoadConst(BasicBlock *bb_, int value, string dest);
00026
00034     virtual void gen_asm(ostream &o) override;
00035
00036 private:
00037     // The constant value to load
00038     int value;
00039
00040     // Name of the destination register or memory variable.
00041     string dest;
00042 };
```

## 6.14 IRInstrMove.h

```
00001 #pragma once
00002
00003 #include <string>
00004 #include <ostream>
00005 #include "../BasicBlock.h"
00006 #include "IRInstrBinaryOp.h"
00007
```

```
00008 using namespace std;
00009
00016 class IRInstrMove : public BaseIRInstr
00017 {
00018 public:
00028     IRInstrMove(BasicBlock *bb_, string src, string dest);
00029
00038     virtual void gen_asm(ostream &o) override;
00039
00040 private:
00041     // The source variable (register or memory location).
00042     string src;
00043
00044     // The destination variable (register or memory location).
00045     string dest;
00046 };
```

## 6.15 IRInstrSet.h

```
00001 #pragma once
00002
00003 #include "BaseIRInstr.h"
00004 #include <ostream>
00005
00014 class IRInstrSet : public BaseIRInstr
00015 {
00016 public:
00025     IRInstrSet(BasicBlock *bb_) : BaseIRInstr(bb_) {};
00026
00038     virtual void gen_asm(std::ostream &o) override;
00039 };
```

## 6.16 IRInstrUnaryOp.h

```
00001 #pragma once
00002
00003 #include "BaseIRInstr.h"
00004
00011 class IRInstrUnaryOp : public BaseIRInstr
00012 {
00013 public:
00023     IRInstrUnaryOp(BasicBlock *bb_, string uniqueOp, string op)
00024         : BaseIRInstr(bb_), uniqueOp(uniqueOp), op(op) {}
00025
00034     virtual void gen_asm(ostream &o);
00035
00036 protected:
00038     string uniqueOp;
00039
00041     string op;
00042
00043 private:
00051     void handleNot(ostream &o);
00052
00060     void handleNeg(ostream &o);
00061
00069     void handleCompl(ostream &o);
00070 };
```

## 6.17 IRVisitor.h

```
00001 #pragma once
00002
00003 #include "antlr4-runtime.h"
00004 #include "generated/ifccBaseVisitor.h"
00005 #include "IR/CFG.h"
00006 #include <map>
00007 #include <string>
00008 #include <vector>
00009
00010 using namespace std;
00011
00019 class IRVisitor : public ifccBaseVisitor
00020 {
00021 public:
```

```
00030         IRVisitor(map<string, int> symbolsTable, int baseStackOffset);
00031
00040         virtual antlrcpp::Any visitProg(ifccParser::ProgContext *ctx) override;
00041
00050         virtual antlrcpp::Any visitReturn_stmt(ifccParser::Return_stmtContext *ctx) override;
00051
00060         virtual antlrcpp::Any visitAssign_stmt(ifccParser::Assign_stmtContext *ctx) override;
00061
00070         virtual antlrcpp::Any visitDecl_stmt(ifccParser::Decl_stmtContext *ctx) override;
00071
00081         antlrcpp::Any visitExpr(ifccParser::ExprContext *expr, bool isFirst);
00082
00091         virtual antlrcpp::Any visitAddsub(ifccParser::AddsubContext *ctx) override;
00092
00101         virtual antlrcpp::Any visitMuldiv(ifccParser::MuldivContext *ctx) override;
00102
00111         virtual antlrcpp::Any visitBitwise(ifccParser::BitwiseContext *ctx) override;
00112
00121         virtual antlrcpp::Any visitComp(ifccParser::CompContext *ctx) override;
00122
00131         virtual antlrcpp::Any visitUnary(ifccParser::UnaryContext *ctx) override;
00132
00141         virtual antlrcpp::Any visitPre(ifccParser::PreContext *ctx) override;
00142
00151         virtual antlrcpp::Any visitPost(ifccParser::PostContext *ctx) override;
00152
00160         void gen_asm(ostream &o);
00161
00169         void setCurrentCFG(CFG *currentCFG);
00170
00178         CFG *getCurrentCFG();
00179
00189         map<string, CFG *> getCFGS();
00190
00191 protected:
00193         map<string, CFG *> cfgs;
00194
00196         CFG *currentCFG;
00197
00198 private:
00207         void assignValueToVar(ifccParser::ExprContext *expr, string varName);
00208
00217         void loadRegisters(ifccParser::ExprContext *leftExpr, ifccParser::ExprContext *rightExpr);
00218
00228         void handleArithmeticOp(ifccParser::ExprContext *leftExpr, ifccParser::ExprContext *rightExpr,
      string op);
00229 };
```

# Index