Laboratory 10: If statement

Grammar

```
Z      ::= E_AS | E_IF.
E_IF   ::= if '(' E_AS ')' '{' E_AS '}' else '{' E_AS '}'
E_AS   ::= E_MDR ('+' | '-' E_MDR)*.
E_MDR  ::= T ('*' | '/' | '%' T)*.
T      ::= i | '(' E_AS ')'.
```

AST node

For the if statement we will create a new AST node that will store the condition expression of the if statement, the expression for the true branch, and the expression for the false branch.

```
class IfStatementAST : public GenericASTNode {
    unique_ptr<GenericASTNode> Cond, TrueExpr, FalseExpr;

public:
    IfStatementAST(
        unique_ptr<GenericASTNode> Cond,
        unique_ptr<GenericASTNode> TrueExpr,
        unique_ptr<GenericASTNode> FalseExpr
    )
    {
        this->Cond = move(Cond);
        this->TrueExpr = move(TrueExpr);
        this->FalseExpr = move(FalseExpr);

        // only if statement
        if (this->FalseExpr == nullptr)
        {
            // ...
        }
    }

    void toString() { return; }
    Value* codegen() { return nullptr; }
};
```

Code generation

When implementing the code generation for the if statements we need to define separate blocks, identified with labels, for each part of the if statement. The `CodeGenTopLevel` will generate by default the `entry block` and we need to create 3 more blocks, `then block`, `else block`, and `merge block`.

To define a new basic block we will use the following code:

```
BasicBlock *myBlock = BasicBlock::Create(*TheContext, "label1");
```

Because the branching instruction needs a boolean value to determine which block to jump to we will create a comparison instruction between our value stored in the `cond` variable and the constant number 0.

After we create the comparison we can create a branch instruction that takes a boolean value and 2 blocks. This instruction will jump to the first block if the condition is true, otherwise will jump to the second block.

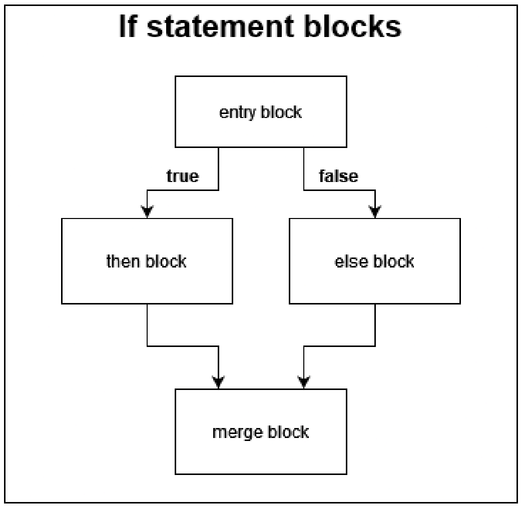

Figure 1: If statement blocks

```
Value* zeroValue =
    ConstantInt::get(*TheContext, APInt(32, 0, true));
compareResult = Builder->CreateICmpNE(cond, zeroValue, "cond");
Builder->CreateCondBr(compareResult, block1, block2);
```

When working with basic blocks we need to identify first in what function we are now. This is provided by the Builder object using the `getParent()` function of the `GetInsertBlock()`.

```
Function *TheFunction = Builder->GetInsertBlock()->getParent();
```

After we obtain the current function we can start working inside a desired block. We will first insert the block at the end of our function, then we need to set the builder to our block and then we can start adding instructions inside the block.

```
// insert the block at the end of the function
TheFunction->insert(TheFunction->end(), MyBlock1);

// set the builder to the desired block
Builder->SetInsertPoint(MyBlock1);

// codegen from other nodes
Value* MyBlock1Result = someNode->codegen();

// jump unconditionally to another block
Builder->CreateBr(MyBlock3);
```

In the end, we want to merge the results from the blocks of our if statement. For this, we will need a new block that will represent the end of our if statement. In this block, we will use a special instruction, the PHI instruction. This will take the value from one or more nodes based on the path of the flow control. To create a PHI instruction we will use the `CreatePHI` function provided by the builder. This function takes as arguments the type of value that will be retrieved, the number of values from the paths (the paths that will merge into the PHI node), and the name of the PHI node value.

```
PHINode *PN =
    Builder->CreatePHI(Type::getInt32Ty(*TheContext), 2, "PHItmp");

PN->addIncoming(MyBlock1Result, MyBlock1);
```

```
PN->addIncoming(MyBlock2Result, MyBlock2);
...
```