Carbonic-C

Asem, Jaffar, Menna, Mosab

Task

The main task was to create an imperative language with the following specifications:

- Static, where object types are fixed at object declarations and cannot change during program execution.
- Three predefined data types: integer, real and boolean.
- Two predefined data structures: records and arrays.
- Full expression syntax with usual set of binary, logical, and comparison operators.
- Compiled, with the target being either an assembly language, of LLVM bitcode, or JVM bytecode, or .NET CIL.
- Program structure: a sequence of data and routine declarations.
- Statements: a standard set (assignment, if/while, return, input/output).

Task

Entity declarations (Variable declarations, Type declarations, Routine declarations)

Types (Predefined[integer, boolean, real], User-defined[record, array])

Statements (Assignment, RoutineCall, WhileLoop, ForLoop, IfStatement)

Loop Utils (range, reverse)

Expressions

Technology

Implementation Language: C++ (g++ 11.3.0)

Lexer Development: Lex (flex 2.6.4)

Parser Development: Bison/Yacc (bison 3.8.2)

Target Platform: LLVM (llvm 14.0.0)

Build Tool: CMake (cmake 2.6.4)

Project Architecture - Lexer

- Technology used: Flex
- Parses input into tokens

```
int getno(){
    return lineno;
}

%}

%%

[ \t]                    ;
[\n]                     {increase_no();}
"var"                    {return carbonic_c::Parser::make_TK_VAR();}
"type"                   {return carbonic_c::Parser::make_TK_TYPE();}
"integer"                {return carbonic_c::Parser::make_TK_INTEGER();}
"real"                   {return carbonic_c::Parser::make_TK_REAL();}
"boolean"                {return carbonic_c::Parser::make_TK_BOOLEAN();}
"array"                  {return carbonic_c::Parser::make_TK_ARRAY();}
"record"                 {return carbonic_c::Parser::make_TK_RECORD();}
"routine"                {return carbonic_c::Parser::make_TK_ROUTINE();}
"return"                 {return carbonic_c::Parser::make_TK_RETURN();}
"is"                     {return carbonic_c::Parser::make_TK_IS();}
```

```
// Define token types
enum class TokenType
{
    /*=========== Declarations Tokens ==============*/
    /*Variables*/
    TK_VAR,
    TK_IS,

    /*Type*/
    TK_TYPE,      You, 2 months ago • [lexer][parser] fin
    TK_INTEGER,
    TK_REAL,
    TK_BOOLEAN,
    TK_ARRAY,
    TK_RECORD,

    /*Routine*/
    TK_ROUTINE,

    /*============= Statement Tokens ==============*/
    /*Loops*/
    TK_WHILE,
    TK_LOOP,
    TK_END,
    TK_FOR,
    TK_FROM,
    TK_IN,
    TK_REVERSE,
    TK_DDOT,
```

Project Architecture - Driver

- Connects Lexer with Parser

```
Asem-Abdelhady, 2 months ago | 1 author (Asem-Abdelhady)
namespace carbonic_c
{
    Driver::Driver() : lexer(*this), parser(lexer, *this) {}

    int Driver::parse_program()
    {
        return parser.parse();
    }
```

```
public:
    Lexer(Driver &driver) : driver(driver) {}
    virtual ~Lexer() {}
    virtual carbonic_c::Parser::symbol_type get_next_token();

private:
    Driver &driver;
};
```

```
static carbonic_c::Parser::symbol_type yylex( carbonic_c::Lexer &lexer , carbonic_c::Driver &driver) {
    return lexer.get_next_token();
}
```

Project Architecture - Parser

- Technology used: Bison/Yacc
- Parses constructs based on the grammar written

```
%start PROGRAM
%%
PROGRAM:
    %empty {}
    | PROGRAM DECLARATION {  program->decls.push_back($2); }
    ;
DECLARATION:
    TYPE_DECL { $$ = $1; }
    | GLOBAL_VAR_DECL { $$ = $1; }
    | ROUTINE_DECL { $$ = $1; }
    ;
ROUTINE_DECL:
    TK_ROUTINE TK_IDENTIFIER TK_LPAREN PARAMETER_LIST TK_RPAREN TK_COLON TYPE TK_IS BODY TK_END { $$ = new ast::RoutineDecl($2, $4, $7, $9); }
    | TK_ROUTINE TK_IDENTIFIER TK_LPAREN PARAMETER_LIST TK_RPAREN TK_IS BODY TK_END { $$ = new ast::RoutineDecl($2, $4, nullptr, $7); }
    | TK_ROUTINE TK_IDENTIFIER TK_LPAREN TK_RPAREN TK_COLON TYPE TK_IS BODY TK_END { $$ = new ast::RoutineDecl($2, nullptr, $6, $8); }
    | TK_ROUTINE TK_IDENTIFIER TK_LPAREN TK_RPAREN TK_IS BODY TK_END { $$ = new ast::RoutineDecl($2, nullptr, nullptr, $6); }
    ;
PARAMETER_LIST:
    PARAMETER_DECL { $$ = new ast::ParameterList(std::vector<ast::ParameterDecl *>()); $$->decls.push_back($1); }
    | PARAMETER_LIST TK_COMMA PARAMETER_DECL{ $1->decls.push_back($3); $$ = $1; }
    ;
PARAMETER_DECL:
    TK_IDENTIFIER TK_COLON TYPE{ $$ = new ast::ParameterDecl($1, $3); }
    ;
BODY:
    %empty { $$ = new ast::Body(std::vector<ast::BodyEntity *>()); }
    | BODY BODY_ENTITY { $1->entities.push_back($2); $$ = $1; }
    ;
```

Project Architecture - Semantics

- Data Structure used: AST
- Checks the validity of the program by utilizing the visitor pattern to go through nodes and check parts of it

```
std::unordered_map<std::string, ast::Type *> typeDeclSymbolTable;
std::unordered_map<std::string, ast::Type *> varDeclSymbolTable;
std::unordered_map<std::string, ast::RoutineDecl *> routineDeclTable;
std::vector<std::pair<std::string, ast::Type *>> varStack;
int routine_vars_n = 0;
ast::Type *actual_type = nullptr;
ast::Type* routine_return_type = nullptr;
ast::Type* current_var_type = nullptr;
void err_second_declaration(std::string name){
    std::cout << "Error: second declaration of " << name << " is invalid.\n";
    exit(1);
}
void err_undefined_obj(std::string obj)
```

```
void Semantic::visitAssignment(ast::Assignment *node)
{
    ast::Type *lhs_type;
    if (node->var)
    {
        node->var->accept(this);
    }
    lhs_type = actual_type;
    ast::Type *rhs_type;
    if (node->expr)
    {
        node->expr->accept(this);
    }
    rhs_type = actual_type;
    typecheck_types(lhs_type, rhs_type);
};
void Semantic::visitRoutineCall(ast::RoutineCall *node
```

Project Architecture - Code Gen.

- Data structure used: AST
- Technology used: LLVM - v14
- Maps the AST of the program to IR code

```
You, 1 second ago | 2 authors (Jaffar Totanji and others)
routine main () : integer is
    var input : integer is 5;
    input := input - 3;
    return input - 2;
end
```

```
; ModuleID = 'Program'
source_filename = "Program"

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

define i32 @main() {
entry:
  %input = alloca i32, align 4
  store i32 5, i32* %input, align 4
  %input1 = load i32, i32* %input, align 4
  %result = sub i32 %input1, 3
  store i32 %result, i32* %input, align 4
  %input2 = load i32, i32* %input, align 4
  %result3 = sub i32 %input2, 2
  ret i32 %result3
}
```

Project Architecture - Interaction

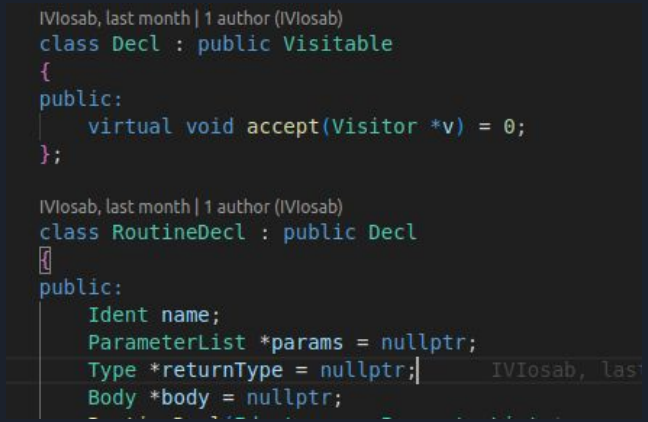
- Build Tool: CMake
- Organizes dependencies, Makes an executable of the compiler, Links LLVM, Enables testing framework (CTest)

```
set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED True)
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR})

find_package(FLEX)
find_package(BISON)
find_package(LLVM REQUIRED CONFIG)

message(STATUS "Found LLVM ${LLVM_PACKAGE_VERSION}")
message(STATUS "Using LLVMConfig.cmake in: ${LLVM_DIR}")

include_directories(${LLVM_INCLUDE_DIRS})
include_directories(${CMAKE_CURRENT_BINARY_DIR})
include_directories(${CMAKE_CURRENT_SOURCE_DIR}/include)

add_definitions(${LLVM_DEFINITIONS})

FLEX_TARGET(CarbonicLexer ${CMAKE_CURRENT_SOURCE_DIR}/lexical_analysis/lexer.l  ${CMAKE_BINARY_DIR}/lexer.cpp)
BISON_TARGET(CarbonicParser ${CMAKE_CURRENT_SOURCE_DIR}/syntax_analysis/parser.ypp ${CMAKE_BINARY_DIR}/parser.cpp)
ADD_FLEX_BISON_DEPENDENCY(CarbonicLexer CarbonicParser)
```

```
# Tests
test: build
    cd build && ctest

# Cleaning
clean:
    rm -rf build
    rm -rf tests/outputs
    rm -f output.txt

.PHONY: build

BUILD_ALL_RUN := $(shell if [ -f build/CMakeCache.txt ]; then echo "1"; else echo "0"; fi)

build:
    ifeq ($(BUILD_ALL_RUN),1)
    $(info build/all has been run before. Running partial build...)
    cd build && make
    else
    $(info build/all has not been run before. Running build/all...)
    $(MAKE) build/all
    endif      Jaffar Totanji, 2 days ago • [fix] .PHONY for build …

build/all: clean
    mkdir -p build
    cd build && cmake .. && make

run: build
    ./build/carbonic_c 2> output.out

run/file: build
    ./build/carbonic_c < input.crbc 2> output.ll
    lli output.ll > output.out
    mv -f output.ll build
```

Major Data Structures - AST

- Class structure
- Visitor design pattern for traversal
- All classes inherit visitable and have their corresponding data



Major Data Structures - IR

- Symbol table for llvm::AllocaInst to load variables and store values in them in IR
- Symbol table for var type to be able to handle equations in their proper type.
  (ex: integer division)
- Visitor design pattern to traverse the ast and map nodes to their corresponding IR code

```
void codeGenerator::visitReturn(ast::Return *node)
{
    if (node->expr)
    {       You, 2 weeks ago • code generation template
        node->expr->accept(this);
    }
    // return input argument
    builder->CreateRet(inferred_value);
};
```

```
llvm::LLVMContext context;
std::unique_ptr<llvm::Module> module;
std::unique_ptr<llvm::IRBuilder<>> builder =
    std::unique_ptr<llvm::IRBuilder<>>(new llvm::IRBuilder<>(context));
llvm::TargetMachine *m_targetMachine;
llvm::Type *inferred_type = nullptr;
llvm::Value *inferred_value = nullptr;
ast::Type *expected_type = nullptr;

llvm::FunctionCallee printf;
```

```
std::unordered_map<std::string, llvm::AllocaInst *> varAllocSymbolTable;
std::unordered_map<std::string, ast::Type *> varType;
```

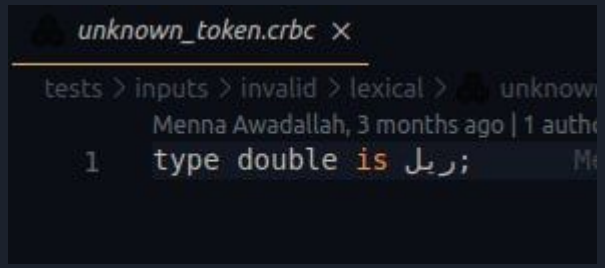
Results - supported

- Routines
- Routine Call
- Return
- Variables (Integer, Real, Boolean, Array, Record)
- Global Variables
- Assignment
- Binary Expressions (*, ÷, +, -, %)
- Comparison Expressions (>, ≥, <, ≤, =, /=)
- Logical Expressions (AND, OR, XOR)
- If Conditions
- While Loops
- For Loops
- Print
- Shadowing
- Recursion

Results - not supported

- For Each (not required)
- Type aliases(not in code generation)
- Type casting
- Array initialization
- Nested access

Example - Invalid token



Example - Syntax error

g.crbc

tests > inputs > invalid > syntax > g.crbc

IVIosab, 2 months ago | 1 author (IVIosab)

```
routine main () : integer is
    var x : integer is 20;
    var y : integer is 15;
    var gcd : integer is 1;

    for i : integer in 1 .. x loop
        if (x % i = 0 and y % i = 0) then
            gcd := i;
        end
    end

    printeger(gcd);

    return 0;
end
```

g.txt

tests > outputs > invalid > syntax > g.txt

```
syntax error, unexpected TK_COLON, expecting TK_IN
Line number: 6
```

Example - Semantic error

bad_return_2.crbc

tests > inputs > invalid > types > bad_routines > bad_return_2.crbc

You, last month | 1 author (You)

```
routine main (): boolean is        You, last month
    var x : integer is 0;

    if true then
        var x : boolean is 1; //shadowing
    end

    return 0;
end
```

bad_return_2.out

tests > outputs > invalid > types > bad_routines > bad_return_2.out

```
1
2  Warning: Shadowing object: x
3  Error: Expected:
4  Boolean,
5  got:
6  Integer
7
```

Example - Fibonacci, recursion

input.crbc M

input.crbc

You, 20 seconds ago | 2 authors (You and others)

```
routine fib (x : integer) : integer is
    if (x = 0) or (x = 1) then
        return 1;
    end
    return fib(x - 1) + fib(x - 2);
end

routine main () : integer is
    print(fib(20));
    return 0;
end
```

You, 20 seconds ago • Uncommitted changes

output.out

output.out

```
1    10946
2
```

Example - Array access

```
routine main () : integer is
    var arr : array [5] integer;
    for i in 0 .. 5 loop
        arr[i] := i;
    end
    for i in 0 .. 5 loop
        print(arr[i]);
    end
    return 0;
end
```

output.out

output.out

```
1   0
2   1
3   2
4   3
5   4
```

Most Notable Contributions

	Lexical Analysis	Driver	Syntax Analysis	Semantic Analysis	Printers	Automation	Code Generation	Testing	VSC Extension
Asem	✔	✔		✔			✔		✔
Jaffar	✔		✔			✔		✔	✔
Menna	✔		✔	✔	✔		✔		
Mosab	✔		✔		✔			✔	✔

Thank you for listening!

Demo Time!