# Language Specification

## Mini-C Compiler

*CS4031 - Compiler Construction | Fall 2025*


### Team Members

Subhan (22K-4316)

Sadiq (22K-4303)

Muhammad Ahmed Haque (22K-4232)

Muhammad Irtiza (22K-4638)

Umer (22K-4160)

Junaid (22K-4369)


*December 2025*

# 1. Language Overview

Mini-C is a statically-typed, imperative programming language designed for numerical computation and pattern generation. It is a carefully designed subset of C, optimized for:

- Mathematical computations (factorial, Fibonacci, arithmetic sequences)
- Array and matrix operations
- Pointer-based data manipulation
- Structured data processing

The language compiles to x86-64 assembly and produces native executables.

# 2. Lexical Specification

## 2.1 Token Categories

| Token Type | Examples | Regex Pattern |
|---|---|---|
| Keywords | int, return, if, while | Reserved words |
| Identifiers | sum, factorial, _count | [a-zA-Z_][a-zA-Z0-9_]* |
| Integer Constants | 42, 100L, 255U | [0-9]+[lLuU]? |
| Float Constants | 3.14, 2.5e-3 | [0-9]*\.[0-9]+ |
| Char Constants | 'a', '\n' | '([^'\\]\|\\...)' |
| String Literals | "hello" | "([^"\\]\|\\...)*" |
| Operators | +, -, *, /, == | See Section 2.3 |
| Delimiters | (, ), {, }, ; | Single characters |

## 2.2 Keywords (Reserved Words)

```
int      long     unsigned   signed    char     double   void
struct   static   extern     return    if       else     while
do       for      break      continue  sizeof
```

## 2.3 Operators

| Category | Operators |
|---|---|
| Arithmetic | + - * / % |
| Relational | < > <= >= == != |
| Logical | && \|\| ! |
| Bitwise | ~ & |
| Assignment | = |
| Pointer | * (dereference) & (address-of) |

| | |
|---|---|
| Member Access | . -> |
| Other | ?: (ternary) sizeof |

# 3. Syntax Specification (BNF/EBNF Grammar)

## 3.1 Program Structure

```
<program> ::= { <declaration> }
<declaration> ::= <variable-declaration>
                | <function-declaration>
                | <struct-declaration>
```

## 3.2 Declarations

```
<variable-declaration> ::= <type-specifier> <declarator> [ "=" <initializer> ] ";"
<function-declaration> ::= <type-specifier> <identifier> "(" <param-list> ")" <com...
<struct-declaration> ::= "struct" <identifier> "{" { <member-decl> } "}" ";"
```

## 3.3 Statements

```
<statement> ::= <compound-statement>
              | <expression-statement>
              | <selection-statement>
              | <iteration-statement>
              | <jump-statement>

<selection-statement> ::= "if" "(" <expression> ")" <statement> [ "else" <statemen...
<iteration-statement> ::= "while" "(" <expression> ")" <statement>
                        | "for" "(" <for-init> [<expr>] ";" [<expr>] ")" <statement>
```

## 3.4 Expressions

```
<expression> ::= <assignment-expression>
<assignment-expression> ::= <conditional-expression>
                          | <unary-expression> "=" <assignment-expression>
<conditional-expression> ::= <logical-or-expr> [ "?" <expr> ":" <conditional-expr> ]
<logical-or-expression> ::= <logical-and-expr> { "||" <logical-and-expr> }
<logical-and-expression> ::= <equality-expr> { "&&" <equality-expr> }
```

# 4. Semantic Rules

## 4.1 Type System

| Type | Size (bytes) | Range/Description |
|---|---|---|
| char | 1 | -128 to 127 |
| unsigned char | 1 | 0 to 255 |
| int | 4 | -2^31 to 2^31-1 |
| unsigned int | 4 | 0 to 2^32-1 |
| long | 8 | -2^63 to 2^63-1 |
| unsigned long | 8 | 0 to 2^64-1 |
| double | 8 | IEEE 754 double precision |
| void | - | No value |
| pointer | 8 | Memory address |

## 4.2 Type Compatibility Rules

- Integer Promotions: char and short are promoted to int in expressions
- Usual Arithmetic Conversions: Operands are converted to a common type
- Pointer Arithmetic: Adding integer to pointer scales by element size
- Array Decay: Arrays decay to pointers in most contexts

## 4.3 Scope Rules

- Block Scope: Variables declared in { } are visible only within that block
- File Scope: Variables/functions declared outside functions are globally visible
- Function Scope: Parameters are visible throughout the function body
- Shadowing: Inner scope declarations shadow outer scope names

# 5. Example Programs

## 5.1 Factorial Computation

```
int factorial(int n) {
    if (n <= 1) return 1;
    return n * factorial(n - 1);
}

int main(void) {
    int result = factorial(5);
    return result;  // Returns 120
}
```

## 5.2 Fibonacci Sequence

```
int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}

int main(void) {
    return fibonacci(10);  // Returns 55
}
```

## 5.3 Array Sum with Pointers

```
int sum_array(int *arr, int size) {
    int total = 0;
    for (int i = 0; i < size; i = i + 1) {
        total = total + arr[i];
    }
    return total;
}

int main(void) {
    int numbers[5] = {1, 2, 3, 4, 5};
    return sum_array(numbers, 5);  // Returns 15
}
```

# 6. Operator Precedence Table

| Precedence | Operator | Associativity |
|---|---|---|
| 1 (highest) | () [] . -> | Left-to-right |
| 2 | ! ~ - * & sizeof | Right-to-left |
| 3 | * / % | Left-to-right |
| 4 | + - | Left-to-right |

| | | |
|---|---|---|
| 5 | < <= > >= | Left-to-right |
| 6 | == != | Left-to-right |
| 7 | && | Left-to-right |
| 8 | \|\| | Left-to-right |
| 9 | ?: | Right-to-left |
| 10 (lowest) | = | Right-to-left |