

# C Language Subset Grammar (Complete Version)

## Terminals

- **Types:** int, char
- **Keywords:** struct, if, else, else if, while, for, return
- **Pointers:** \*, &
- **Punctuation:** {, }, ;, (, ), =, ,, <, >, #, [, ]
- **Operators:**
  - Arithmetic: +, -, \*, /, %, ++, --
  - Assignment: =, +=, -=, \*=, /=, %=
  - Relational: ==, !=, <, >, <=, >=
  - Logical: &&, ||, !
- **Preprocessor:** #include
- **Comments:**
  - Line comment: //
  - Block comment: /\*, \*/

## Non-Terminals (Part 1)

$Program \rightarrow (Includes|Comment) GlobalDeclarations Functions MainFunction$   
 $Includes \rightarrow (Include|Comment) Includes \mid \epsilon$   
 $Include \rightarrow \#include < HeaderName > \mid \#include " HeaderName "$   
 $HeaderName \rightarrow [a - zA - Z0 - 9.]+$   
 $Comment \rightarrow LineComment \mid BlockComment$   
 $LineComment \rightarrow // TextUntilNewline$   
 $BlockComment \rightarrow /* TextUntilEndComment */$   
 $TextUntilNewline \rightarrow [\backslash n]^*$   
 $TextUntilEndComment \rightarrow ([^*]|* + [^*/])^* **$   
 $MainFunction \rightarrow \text{int main} ( ) \{ Body \}$   
 $Functions \rightarrow (Function|Comment) Functions \mid \epsilon$   
 $Function \rightarrow Type Identifier ( Parameters ) \{ Body \}$   
 $Type \rightarrow \text{int} \mid \text{char} \mid \text{void} \mid \text{struct Identifier}$   
 $GlobalDeclarations \rightarrow (GlobalDecl|Comment) GlobalDeclarations \mid \epsilon$   
 $GlobalDecl \rightarrow Type GlobalVarDecl ;$   
 $GlobalVarDecl \rightarrow Identifier \mid * Identifier \mid Identifier [ ] \mid Identifier = Literal$   
 $Body \rightarrow LocalDeclarations Statements$   
 $LocalDeclarations \rightarrow (LocalDecl|Comment) LocalDeclarations \mid \epsilon$   
 $LocalDecl \rightarrow Type VarDecl ;$   
 $VarDecl \rightarrow Identifier \mid * Identifier \mid Identifier [ Integer ] \mid Identifier = Expression$   
 $Statements \rightarrow (Statement|Comment) Statements \mid \epsilon$

## Non-Terminals (Part 2)

$Statement \rightarrow \text{if} ( Expression ) \{ Statements \} Else$   
 $\quad | \text{while} ( Expression ) \{ Statements \}$   
 $\quad | \text{for} ( ForInit ; Expression ; ForUpdate ) \{ Statements \}$   
 $\quad | ExpressionStmt$   
 $\quad | \text{printf} ( FormatString , Arguments ) ;$   
 $\quad | \text{return} ReturnExpr ;$   
 $ForInit \rightarrow AssignExpr \mid \epsilon$   
 $ForUpdate \rightarrow AssignExpr \mid \epsilon$   
 $ExpressionStmt \rightarrow Expression ;$   
 $Else \rightarrow \text{else} \{ Statements \} \mid \text{else if} ( Expression ) \{ Statements \} Else \mid \epsilon$   
 $Expression \rightarrow AssignExpr$   
 $AssignExpr \rightarrow LogicalOrExpr \mid UnaryExpr AssignOp AssignExpr$   
 $AssignOp \rightarrow = \mid += \mid -= \mid *= \mid /= \mid \%=$   
 $LogicalOrExpr \rightarrow LogicalAndExpr \mid \mid LogicalOrExpr \mid LogicalAndExpr$   
 $LogicalAndExpr \rightarrow EqualityExpr \&\& LogicalAndExpr \mid EqualityExpr$   
 $EqualityExpr \rightarrow RelationalExpr EqualityOp EqualityExpr \mid RelationalExpr$   
 $EqualityOp \rightarrow == \mid !=$   
 $RelationalExpr \rightarrow AdditiveExpr RelationalOp RelationalExpr \mid AdditiveExpr$   
 $RelationalOp \rightarrow < \mid > \mid <= \mid >=$   
 $AdditiveExpr \rightarrow MultiplicativeExpr AdditiveOp AdditiveExpr \mid MultiplicativeExpr$   
 $AdditiveOp \rightarrow + \mid -$   
 $MultiplicativeExpr \rightarrow UnaryExpr MultiplicativeOp MultiplicativeExpr \mid UnaryExpr$   
 $MultiplicativeOp \rightarrow * \mid / \mid \%$   
 $UnaryExpr \rightarrow PostfixExpr \mid UnaryOp UnaryExpr$   
 $UnaryOp \rightarrow + \mid - \mid ++ \mid -- \mid ! \mid *$   
 $PostfixExpr \rightarrow PrimaryExpr \mid PostfixExpr ++ \mid PostfixExpr --$   
 $PrimaryExpr \rightarrow Identifier \mid Literal \mid StringLit \mid ( Expression )$   
 $FunctionCall \rightarrow Identifier ( Arguments )$   
 $Arguments \rightarrow Expression , Arguments \mid Expression \mid \epsilon$   
 $ReturnExpr \rightarrow Expression \mid \epsilon$   
 $Identifier \rightarrow [a - zA - Z\_][a - zA - Z0 - 9]*$   
 $Literal \rightarrow Integer \mid CharLit$   
 $Integer \rightarrow [0 - 9]^+$   
 $CharLit \rightarrow '.'$   
 $StringLit \rightarrow \".*\"$   
 $FormatString \rightarrow \"(. * \% [dsc] ). *\"$

## Complete Example Program

```
#include <stdio.h> // Standard I/O header

/*
 * Global variables section
 * Multi-line comment
 */
char* message = "Hello World";
int numbers[5] = {1, 2, 3, 4, 5};

// Function to calculate factorial recursively
int factorial(int n) {
    // Base case
    if (n <= 1) return 1;

    /* Recursive case */
    return n * factorial(n - 1);
}

int main() {
    char c = 'A'; // Character variable
    int x = 5;
    int* ptr = &x; // Pointer to x

    printf("%s\n", message); // Print message

    // Calculate and print factorial
    printf("Factorial of %d is %d\n", x, factorial(x));

    /* Print array elements */
    for (int i = 0; i < 5; i++) {
        printf("%d ", numbers[i]);
    }

    return 0; // Successful execution
}
```