EBNF for C with OpenMP

TranslationUnit ::= (ElementsOfTranslation)+

ElementsOfTranslation ::= ExternalDeclaration
| UnknownCpp
| UnknownPragma

ExternalDeclaration ::= Declaration
| FunctionDefinition
| DeclareReductionDirective
| ThreadPrivateDirective

FunctionDefinition ::= (DeclarationSpecifiers)? Declarator (DeclarationList)? CompoundStatement

Declaration ::= DeclarationSpecifiers (InitDeclaratorList)? ";"

DeclarationList ::= (Declaration)+

DeclarationSpecifiers ::= (ADeclarationSpecifier)+

ADeclarationSpecifier ::= StorageClassSpecifier
| TypeSpecifier
| TypeQualifier

StorageClassSpecifier ::= <AUTO> | <REGISTER> | <STATIC> | <EXTERN> | <TYPEDEF>

TypeSpecifier ::= <VOID> | <CHAR> | <SHORT> | <INT> | <LONG> | <FLOAT> | <DOUBLE> | <SIGNED> | <UNSIGNED> | StructOrUnionSpecifier | EnumSpecifier | TypedefName

TypeQualifier ::= <RESTRICT> | <CONST> | <VOLATILE> | <INLINE> | <CCONST> | <CINLINED> | <CINLINED2> | <CSIGNED> | <CSIGNED2>

StructOrUnionSpecifier ::= (StructOrUnionSpecifierWithList | StructOrUnionSpecifierWithId)

StructOrUnionSpecifierWithList ::= StructOrUnion (<IDENTIFIER>)? "{" StructDeclarationList "}"

StructOrUnionSpecifierWithId ::= StructOrUnion <IDENTIFIER>

StructOrUnion ::= <STRUCT> | <UNION>

StructDeclarationList ::= (StructDeclaration)+

InitDeclaratorList ::= InitDeclarator ("," InitDeclarator)*

InitDeclarator ::= Declarator ("=" Initializer)?

StructDeclaration ::= SpecifierQualifierList StructDeclaratorList ";"

SpecifierQualifierList ::= (ASpecifierQualifier)+

ASpecifierQualifier ::= TypeSpecifier
| TypeQualifier

StructDeclaratorList ::= StructDeclarator ("," StructDeclarator)*

StructDeclarator ::= StructDeclaratorWithDeclarator
| StructDeclaratorWithBitField

StructDeclaratorWithDeclarator ::= Declarator (":" ConstantExpression)?

StructDeclaratorWithBitField ::= ":" ConstantExpression

EnumSpecifier ::= EnumSpecifierWithList
| EnumSpecifierWithId

EnumSpecifierWithList ::= <ENUM> (<IDENTIFIER>)? "{" EnumeratorList "}"

EnumSpecifierWithId ::= <ENUM> <IDENTIFIER>

EnumeratorList ::= Enumerator ("," Enumerator)*

```
Enumerator ::= <IDENTIFIER> ( "=" ConstantExpression )?
Declarator ::= ( Pointer )? DirectDeclarator
DirectDeclarator ::= IdentifierOrDeclarator DeclaratorOpList
DeclaratorOpList ::= ( ADeclaratorOp )*
ADeclaratorOp ::= DimensionSize
                | ParameterTypeListClosed
                | OldParameterListClosed
DimensionSize ::= "[" ( ConstantExpression )? "]"
ParameterTypeListClosed ::= "(" ( ParameterTypeList )? ")"
OldParameterListClosed ::= "(" ( OldParameterList )? ")"
IdentifierOrDeclarator ::= <IDENTIFIER>
                         | "(" Declarator ")"
Pointer ::= ( "*" | "^" ) ( TypeQualifierList )? ( Pointer )?
TypeQualifierList ::= ( TypeQualifier )+
ParameterTypeList ::= ParameterList ( "," "..." )?
ParameterList ::= ParameterDeclaration ( "," ParameterDeclaration )*
ParameterDeclaration ::= DeclarationSpecifiers ParameterAbstraction
ParameterAbstraction ::= Declarator
                       | AbstractOptionalDeclarator
AbstractOptionalDeclarator ::= ( AbstractDeclarator )?
OldParameterList ::= <IDENTIFIER> ( "," <IDENTIFIER> )*
Initializer ::= AssignmentExpression
              | ArrayInitializer
ArrayInitializer ::= "{" InitializerList ( "," )? "}"
InitializerList ::= Initializer ( "," Initializer )*
TypeName ::= SpecifierQualifierList ( AbstractDeclarator )?
AbstractDeclarator ::= AbstractDeclaratorWithPointer
                     | DirectAbstractDeclarator
AbstractDeclaratorWithPointer ::= Pointer ( DirectAbstractDeclarator )?
DirectAbstractDeclarator ::= AbstractDimensionOrParameter DimensionOrParameterList
AbstractDimensionOrParameter ::= AbstractDeclaratorClosed
                               | DimensionSize
                               | ParameterTypeListClosed
AbstractDeclaratorClosed ::= "(" AbstractDeclarator ")"
DimensionOrParameterList ::= ( ADimensionOrParameter )*
ADimensionOrParameter ::= DimensionSize
                        | ParameterTypeListClosed
TypedefName ::= <IDENTIFIER>
Statement ::= LabeledStatement
            | ExpressionStatement
            | CallStatement
```

| CompoundStatement

| SelectionStatement

| IterationStatement

| JumpStatement

| UnknownPragma

| OmpConstruct

| OmpDirective

| UnknownCpp

UnknownCpp ::= "#" <UNKNOWN_CPP>

OmpEol ::= <OMP_CR>

| <OMP_NL>

OmpConstruct ::= ParallelConstruct

| ForConstruct

| SectionsConstruct

| SingleConstruct

| ParallelForConstruct

| ParallelSectionsConstruct

| TaskConstruct

| MasterConstruct

| CriticalConstruct

| AtomicConstruct

| OrderedConstruct

OmpDirective ::= BarrierDirective

| TaskwaitDirective

| TaskyieldDirective

| FlushDirective

ParallelConstruct ::= OmpPragma ParallelDirective Statement

OmpPragma ::= "#" <PRAGMA> <OMP>

UnknownPragma ::= "#" <PRAGMA> <UNKNOWN_CPP>

ParallelDirective ::= <PARALLEL> UniqueParallelOrDataClauseList OmpEol

UniqueParallelOrDataClauseList ::= (AUniqueParallelOrDataClause)*

AUniqueParallelOrDataClause ::= UniqueParallelClause

| DataClause

UniqueParallelClause ::= IfClause

| NumThreadsClause

IfClause ::= <IF> "(" Expression ")"

NumThreadsClause ::= <NUM_THREADS> "(" Expression ")"

DataClause ::= OmpPrivateClause

| OmpFirstPrivateClause

| OmpLastPrivateClause

| OmpSharedClause

| OmpCopyinClause

| OmpDfltSharedClause

| OmpDfltNoneClause

| OmpReductionClause

OmpPrivateClause ::= <PRIVATE> "(" VariableList ")"

OmpFirstPrivateClause ::= <FIRSTPRIVATE> "(" VariableList ")"

OmpLastPrivateClause ::= <LASTPRIVATE> "(" VariableList ")"

OmpSharedClause ::= <SHARED> "(" VariableList ")"

OmpCopyinClause ::= <COPYIN> "(" VariableList ")"

OmpDfltSharedClause ::= <DFLT> "(" <SHARED> ")"

OmpDfltNoneClause ::= <DFLT> "(" <NONE> ")"

OmpReductionClause ::= <REDUCTION> "(" ReductionOp ":" VariableList ")"

ForConstruct ::= OmpPragma ForDirective OmpForHeader Statement

ForDirective ::= <FOR> UniqueForOrDataOrNowaitClauseList OmpEol

UniqueForOrDataOrNowaitClauseList ::= (AUniqueForOrDataOrNowaitClause)*

AUniqueForOrDataOrNowaitClause ::= UniqueForClause

| DataClause

| NowaitClause

NowaitClause ::= <NOWAIT>

UniqueForClause ::= <ORDERED>

| UniqueForClauseSchedule

| UniqueForCollapse

UniqueForCollapse ::= <COLLAPSE> "(" Expression ")"

UniqueForClauseSchedule ::= <SCHEDULE> "(" ScheduleKind ("," Expression)? ")"

ScheduleKind ::= <STATIC> | <DYNAMIC> | <GUIDED> | <RUNTIME>

OmpForHeader ::= <FOR> "(" OmpForInitExpression ";" OmpForCondition ";" OmpForReinitExpression ")"

OmpForInitExpression ::= <IDENTIFIER> "=" Expression

OmpForCondition ::= OmpForLTCondition

| OmpForLECondition

| OmpForGTCondition

| OmpForGECondition

OmpForLTCondition ::= <IDENTIFIER> "<" Expression

OmpForLECondition ::= <IDENTIFIER> "<=" Expression

OmpForGTCondition ::= <IDENTIFIER> ">" Expression

OmpForGECondition ::= <IDENTIFIER> ">=" Expression

OmpForReinitExpression ::= PostIncrementId

| PostDecrementId

| PreIncrementId

| PreDecrementId

| ShortAssignPlus

| ShortAssignMinus

| | OmpForAdditive
| | OmpForSubtractive
| | OmpForMultiplicative
PostIncrementId ::= <IDENTIFIER> "++"
PostDecrementId ::= <IDENTIFIER> "--"
PreIncrementId ::= "++" <IDENTIFIER>
PreDecrementId ::= "--" <IDENTIFIER>
ShortAssignPlus ::= <IDENTIFIER> "+=" Expression
ShortAssignMinus ::= <IDENTIFIER> "-=" Expression
OmpForAdditive ::= <IDENTIFIER> "=" <IDENTIFIER> "+" AdditiveExpression
OmpForSubtractive ::= <IDENTIFIER> "=" <IDENTIFIER> "-" AdditiveExpression
OmpForMultiplicative ::= <IDENTIFIER> "=" MultiplicativeExpression "+" <IDENTIFIER>
SectionsConstruct ::= OmpPragma <SECTIONS> NowaitDataClauseList OmpEol SectionsScope
NowaitDataClauseList ::= (ANowaitDataClause)*
ANowaitDataClause ::= NowaitClause
| | DataClause
SectionsScope ::= "{" (Statement)? (ASection)* "}"
ASection ::= OmpPragma <SECTION> OmpEol Statement
SingleConstruct ::= OmpPragma <SINGLE> SingleClauseList OmpEol Statement
SingleClauseList ::= (ASingleClause)*
ASingleClause ::= NowaitClause
| | DataClause
| | OmpCopyPrivateClause
OmpCopyPrivateClause ::= <COPYPRIVATE> "(" VariableList ")"
TaskConstruct ::= OmpPragma <TASK> (TaskClause)* OmpEol Statement
TaskClause ::= DataClause
| | UniqueTaskClause
UniqueTaskClause ::= IfClause
| | FinalClause
| | UntiedClause
| | MergeableClause
FinalClause ::= <FINAL> "(" Expression ")"
UntiedClause ::= <UNTIED>
MergeableClause ::= <MERGEABLE>
ParallelForConstruct ::= OmpPragma <PARALLEL> <FOR> UniqueParallelOrUniqueForOrDataClauseList OmpEol OmpForHeader Statement
UniqueParallelOrUniqueForOrDataClauseList ::= (AUniqueParallelOrUniqueForOrDataClause)*
AUniqueParallelOrUniqueForOrDataClause ::= UniqueParallelClause
| | UniqueForClause
| | DataClause
ParallelSectionsConstruct ::= OmpPragma <PARALLEL> <SECTIONS> UniqueParallelOrDataClauseList OmpEol SectionsScope

MasterConstruct ::= OmpPragma <MASTER> OmpEol Statement

CriticalConstruct ::= OmpPragma <CRITICAL> (RegionPhrase)? OmpEol Statement

RegionPhrase ::= "(" <IDENTIFIER> ")"

AtomicConstruct ::= OmpPragma <ATOMIC> (AtomicClause)? OmpEol Statement

AtomicClause ::= <READ> | <WRITE> | <UPDATE> | <CAPTURE>

FlushDirective ::= OmpPragma <FLUSH> (FlushVars)? OmpEol

FlushVars ::= "(" VariableList ")"

OrderedConstruct ::= OmpPragma <ORDERED> OmpEol Statement

BarrierDirective ::= OmpPragma <BARRIER> OmpEol

TaskwaitDirective ::= OmpPragma <TASKWAIT> OmpEol

TaskyieldDirective ::= OmpPragma <TASKYIELD> OmpEol

ThreadPrivateDirective ::= OmpPragma <THREADPRIVATE> "(" VariableList ")" OmpEol

DeclareReductionDirective ::= OmpPragma <DECLARE> <REDUCTION> "(" ReductionOp ":" ReductionTypeList ":" Expression ")" (InitializerClause)? OmpEol

ReductionTypeList ::= (TypeSpecifier)*

InitializerClause ::= AssignInitializerClause
| ArgumentInitializerClause

AssignInitializerClause ::= <INITIALIZER> "(" <IDENTIFIER> "=" Initializer ")"

ArgumentInitializerClause ::= <INITIALIZER> "(" <IDENTIFIER> "(" ExpressionList ")" ")"

ReductionOp ::= <IDENTIFIER> | "+" | "*" | "-" | "&" | "^" | "|" | "||" | "&&"

VariableList ::= <IDENTIFIER> ("," <IDENTIFIER>)*

LabeledStatement ::= SimpleLabeledStatement
| CaseLabeledStatement
| DefaultLabeledStatement

SimpleLabeledStatement ::= <IDENTIFIER> ":" Statement

CaseLabeledStatement ::= <CASE> ConstantExpression ":" Statement

DefaultLabeledStatement ::= <DFLT> ":" Statement

ExpressionStatement ::= (Expression)? ";"

CompoundStatement ::= "{" (CompoundStatementElement)* "}"

CompoundStatementElement ::= Declaration
| Statement

SelectionStatement ::= IfStatement
| SwitchStatement

IfStatement ::= <IF> "(" Expression ")" Statement (<ELSE> Statement)?

SwitchStatement ::= <SWITCH> "(" Expression ")" Statement

IterationStatement ::= WhileStatement
| DoStatement
| ForStatement

WhileStatement ::= <WHILE> "(" Expression ")" Statement

DoStatement ::= <DO> Statement <WHILE> "(" Expression ")" ";"

ForStatement ::= <FOR> "(" (Expression)? ";" (Expression)? ";" (Expression)? ")" Statement

```
JumpStatement ::= GotoStatement
                | ContinueStatement
                | BreakStatement
                | ReturnStatement
GotoStatement ::= <GOTO> <IDENTIFIER> ";"
ContinueStatement ::= <CONTINUE> ";"
BreakStatement ::= <BREAK> ";"
ReturnStatement ::= <RETURN> ( Expression )? ";"
Expression ::= AssignmentExpression ( "," AssignmentExpression )*
AssignmentExpression ::= NonConditionalExpression
                       | ConditionalExpression
NonConditionalExpression ::= UnaryExpression AssignmentOperator AssignmentExpression
AssignmentOperator ::= "=" | "*=" | "/=" | "%=" | "+=" | "-=" | "<<=" | ">>=" | "&=" | "^=" | "|="
ConditionalExpression ::= LogicalORExpression ( "?" Expression ":" ConditionalExpression )?
ConstantExpression ::= ConditionalExpression
LogicalORExpression ::= LogicalANDExpression ( "||" LogicalORExpression )?
LogicalANDExpression ::= InclusiveORExpression ( "&&" LogicalANDExpression )?
InclusiveORExpression ::= ExclusiveORExpression ( "|" InclusiveORExpression )?
ExclusiveORExpression ::= ANDExpression ( "^" ExclusiveORExpression )?
ANDExpression ::= EqualityExpression ( "&" ANDExpression )?
EqualityExpression ::= RelationalExpression ( EqualOptionalExpression )?
EqualOptionalExpression ::= EqualExpression
                          | NonEqualExpression
EqualExpression ::= "==" EqualityExpression
NonEqualExpression ::= "!=" EqualityExpression
RelationalExpression ::= ShiftExpression ( RelationalOptionalExpression )?
RelationalOptionalExpression ::= RelationalLTExpression
                               | RelationalGTExpression
                               | RelationalLEExpression
                               | RelationalGEExpression
RelationalLTExpression ::= "<" RelationalExpression
RelationalGTExpression ::= ">" RelationalExpression
RelationalLEExpression ::= "<=" RelationalExpression
RelationalGEExpression ::= ">=" RelationalExpression
ShiftExpression ::= AdditiveExpression ( ShiftOptionalExpression )?
ShiftOptionalExpression ::= ShiftLeftExpression
                          | ShiftRightExpression
ShiftLeftExpression ::= ">>" ShiftExpression
ShiftRightExpression ::= "<<" ShiftExpression
AdditiveExpression ::= MultiplicativeExpression ( AdditiveOptionalExpression )?
AdditiveOptionalExpression ::= AdditivePlusExpression
```

| AdditiveMinusExpression

AdditivePlusExpression ::= "+" AdditiveExpression

AdditiveMinusExpression ::= "-" AdditiveExpression

MultiplicativeExpression ::= CastExpression (MultiplicativeOptionalExpression)?

MultiplicativeOptionalExpression ::= MultiplicativeMultiExpression

| MultiplicativeDivExpression

| MultiplicativeModExpression

MultiplicativeMultiExpression ::= "*" MultiplicativeExpression

MultiplicativeDivExpression ::= "/" MultiplicativeExpression

MultiplicativeModExpression ::= "%" MultiplicativeExpression

CastExpression ::= CastExpressionTyped

| UnaryExpression

CastExpressionTyped ::= "(" TypeName ")" CastExpression

UnaryExpression ::= UnaryExpressionPreIncrement

| UnaryExpressionPreDecrement

| UnarySizeofExpression

| UnaryCastExpression

| PostfixExpression

UnaryExpressionPreIncrement ::= "++" UnaryExpression

UnaryExpressionPreDecrement ::= "--" UnaryExpression

UnaryCastExpression ::= UnaryOperator CastExpression

UnarySizeofExpression ::= SizeofTypeName

| SizeofUnaryExpression

SizeofUnaryExpression ::= <SIZEOF> UnaryExpression

SizeofTypeName ::= <SIZEOF> "(" TypeName ")"

UnaryOperator ::= "&" | "*" | "+" | "-" | "~" | "!"

PostfixExpression ::= PrimaryExpression PostfixOperationsList

PostfixOperationsList ::= (APostixOperation)*

APostixOperation ::= BracketExpression

| ArgumentList

| DotId

| ArrowId

| PlusPlus

| MinusMinus

PlusPlus ::= "++"

MinusMinus ::= "--"

BracketExpression ::= "[" Expression "]"

ArgumentList ::= "(" (ExpressionList)? ")"

DotId ::= "." <IDENTIFIER>

ArrowId ::= "->" <IDENTIFIER>

PrimaryExpression ::= <IDENTIFIER>

| | Constant

| | ExpressionClosed

ExpressionClosed ::= "(" Expression ")"

ExpressionList ::= AssignmentExpression ("," AssignmentExpression)*

Constant ::= <INTEGER_LITERAL> | <FLOATING_POINT_LITERAL> | <CHARACTER_LITERAL> | (<STRING_LITERAL>)+

BeginNode

EndNode

DummyFlushDirective

SimplePrimaryExpression

CallStatement

PreCallNode

PostCallNode