APPENDIX E
BNF Syntax for Turbo Modula-2

The syntax of the Turbo Modula-2 language is presented here using the formalism known as Backus-Naur Form (BNF). The following symbols are meta symbols belonging to the BNF formalism; they are not symbols of the language.

<term>	Names of language constructs are surrounded by "<" and ">".
{ X }*	Represents zero or more repetitions of X.
[X]	Means X is optional.
X	Means X is mandatory.
X | Y	Indicates that X and Y are alternatives and that either X or Y must be used.
"X" or 'X'	Means X is written exactly as shown.
(X)	Means X must be chosen.

All other symbols are part of the language (reserved words of Turbo Modula-2 are in **boldface** type). For easy reference, the syntactic constructs are listed alphabetically.

```
<ActualParameters> ::= "(" [<ExpList> ] ")"

<AddOperator> ::= "+" | "-" | "OR"

<ArrayType> ::= "ARRAY" <SimpleType> {"," <SimpleType>}*
"OF"
<type>

<assignment> ::= <designator> ":=" <expression>

<block> ::= { <declaration> }* [ "BEGIN" <StatementSequence> ]
            [<ExceptionHandler> ] "END"

<case> ::= [ <CaseLabelList> ":" <StatementSequence> ]
```

```
<CaseLabelList> ::= <CaseLabels> {"," <CaseLabels>}*

<CaseLabels> ::= <ConstExpression> [ ".." <ConstExpression> ]

<CaseStatement> :: = "CASE" <expression> "OF" <case>
                     { "|" [ <case> ] }
                     [ "ELSE" <StatementSequence> ]
                       "END"

<character> ::= <letter> | <digit> | " " | "!" | "'" | "#" |
                "$" | "%" | "&" | "'" | "(" | ")" | "*" | "+" |
                "," | "-" | "." | "/" | ":" | ";" | "<" | "=" |
                ">" | "?" | "@" | "[" | "\" | "]" | "^" | "_" |
                "`" | "{" | "|" | "}" | "~"

<CompilationUnit> ::= <DefinitionModule> | ["IMPLEMENTATION"]
                        <ProgramModule>

<ConstantDeclaration> ::= <ident> "=" <ConstExpression>

<ConstExpression> ::= <expression>

<declaration> ::= "CONST" { <ConstantDeclaration> ";" }* |
                  "TYPE"  { <TypeDeclaration>     ";" }* |
                  "VAR "  { <VariableDeclaration> ";" }* |
                  <ExceptionDeclaration>          ";" |
                  <ProcedureDeclaration>          ";" |
                  <ModuleDeclaration>             ";"

<definition> ::=  "CONST" { <ConstantDeclaration> ";" }* |
                  "TYPE"  { <ident> ["="<type>] ";" }* |
                  "VAR "  { <VariableDeclaration> ";" }* |
                  <ExceptionDeclaraton>           ";" |
                  <ProcedureHeading>              ";"

<DefinitionModule> ::= "DEFINITION MODULE" <ident> ";"
                       { <import> }* { <definition> }* "END"
                       <ident> "."
```

```
<designator> ::= qualident { "." <ident>|
"["ExpList"]"| " ^ "}*

<digit> ::= <octalDigit> | "8" | "9"

<element> ::= <expression> [ ".." <expression> ]

<enumeration> ::= "(" <IdentList> ")"

<exception> ::= [ <IdentList> ":" <StatementSequence> ]

<export> ::= "EXPORT" [ "QUALIFIED" ] <IdentList> ";"

<expression> ::= <SimpleExpression> { <relation>
<SimpleExpression> }*

<ExceptionDeclaration>::="EXCEPTION"<ident> {"," <ident>}*

<ExceptionHandler> ::="EXCEPTION" <exception> { "|"
                      <exception> }*
                      [ "ELSE" <StatementSequence> ]

<ExpList> ::= <expression> { "," <expression> }*

<factor> ::=<number> | <string> | <set> | <designator>
[<ActualParamters>] |
            "(" <expresion> ")" | "NOT" <factor>

<FieldList> ::=[ <IdentList> ":" <type> |
                 "CASE" [<ident>] ":" <qualident>
                 "OF" <variant> {"|" <variant>}*
                 [ "ELSE" <FieldListSequence> ] "END" ]

<FieldListSequence> ::= <FieldList> {";" <FieldList>}*

<ForStatement> ::="FOR" <ident> ":=" <expression> "TO"
                    <expression>
                    [ "BY" <ConstExpression> ]
                    "DO" <StatementSequence> "END"
```

```
<FormalParamenters> ::= "(" [ <FPSection>
                            { ";" <FPSection> }*]")"
                            [ ":" <qualident> ]

<FormalType> ::= [ "ARRAY OF" <qualident> ]

<FormalTypeList> ::="(" [ ["VAR"] <FormalType>
                      {"," [VAR"] <FormalType>}* ] ")" [ ":"
                      <qualident> ]

<FPSection> ::= [ "VAR" ] <IdentList> ":" <FormalType>

<hexDigit> ::= <digit> | "A" | "B" | "C" | "D" | "E" | "F"

<ident> ::= letter {letter | <digit>}*

<IdentList> ::= <ident> {"," <ident>}*

<IfStatement> ::="IF" <expression> "THEN"
                 <StatementSequence>
                 {"ELSIF" <expression> "THEN"
                 <StatementSequence> }*
                 ["ELSE <StatementSequence> ]
                 "END"

<import> ::= [ "FROM" <ident> ] "IMPORT" <IdentList> ";"

<InlineCode> ::= "CODE" "(" <string> ")" "END"

<integer> ::=<digit> {<digit>}* |
             {<octalDigit>}* ( "B" | "C" ) |
              <digit> {<hexDigit>}*

<letter> ::="A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" |
            "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" |
            "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" |
            "Y" | "Z" | "a" | "b" | "c" | "d" | "e" | "f" |
            "g" | "h" | "i" | "j" | "k" | "l | "m" | "n" |
            "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" |
            "w" | "x" | "y" | "z"
```

```
<LoopStatement> ::= "LOOP" <StatementSequence> "END"

<ModuleDeclaration> ::="MODULE" <ident> [ <priority> ] ";"
                       { <import> } [ <export> ] <block>
                       <ident>

<MulOperator> ::= "*" | "/" | "DIV" | "MOD" | "AND"

<number> ::= <integer> | <real>

<octalDigit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7"

<PointerType> ::= "POINTER TO" <type>

<priority> ::= "[" <ConstExpression> "]"

<ProcedureCall> ::= <designator> [ <ActualParameters> ]

<ProcedureDeclaration> ::= <ProcedureHeading> ";"
                           ( <block> | <InlineCode> )
                           <ident>

<ProcedureHeading> ::= "PROCEDURE" <ident>
[ <FormalParameters> ]

<ProcedureType> ::= "PROCEDURE" [FormalTypeList]

<ProgramModule> ::="MODULE" <ident> [ <priority> ] ";"
                    { <import> }* <block> <ident> "."

<qualident> ::= <ident> {"." <ident>}*

<RaiseStatement> ::= "RAISE" [ <ident> [ "," <expression> ] ]

<real> ::= <digit> {<digit>}* "." {<digit>}* [ ScaleFactor ]

<RecordType> ::= "RECORD" <FieldListSequence> "END"

<relation> ::= "=" | ">" | "<" | ">=" | "<=" | "#" | "< >"
| "IN"
```

```
<RepeatStatement> ::= "REPEAT <StatementSequence> "UNTIL"
<expression>

<ScaleFactor> ::= ( "E" | "D" ) [ "+" | "-" ] <digit>
{<digit>}*

<set> ::= [ <qualident> ] "{" [ <element> { ","
<element> }* ] "}"

<SetType> ::= "SET OF" <SimpleType>

<SimpleExpression> ::= [ "+" | "-" ] <term> { <AddOperator>
<term> }*

<SimpleType> ::= <qualident> | <enumeration> |
<SubrangeType>

<statement>::=[ assignment | ProcedureCall | IfStatement |
               CaseStatement | WhileStatement | RepeatStatement |
               LoopStatement | ForStatement | WithStatement |
               RaiseStatement | "EXIT" | "RETURN" [expression] ]

<StatementSequence> ::= <statement> { ";" <statement> }*

<string> ::= "'" {<character>}* "'" | '"' {<character>}* '"'

<SubrangeType> ::=[ <qualident> ] "[" <ConstExpression> ".."
                   <ConstExpression> "]"

<term> ::= <factor> { <MulOperator> <factor> }*

<type> ::=<SimpleType> | <ArrayType> | <RecordType> |
          <SetType> | <PointerType> | <ProcedureType>

<TypeDeclaration> ::= <ident> "=" <type>
```

```
<VariableDeclaration>::= <ident>["[" <ConstExpression>"]"]
                         { "," <ident> [ "[" <ConstExpression>
                         "]" ] }*
                         ":" <type>

<variant> ::= [ <CaseLabelList> ":" <FieldListSequence> ]

<WhileStatement> ::="WHILE" <expression> "DO"
                     <StatementSequence>
                     "END"

<WithStatement> ::="WITH" <designator> "DO"
                   <StatementSequence>
                   "END"
```