Oberon-07M
User Guide

1. Introduction

This guide describes compiler and linker for building 32-bits applications for operation system Windows from sources in Oberon-07M programming language. Oberon-07M is slightly modified Oberon-07 programming language, which described in [1].

2. Differences between Oberon-07M and Oberon-07

1) Identifiers can contain "_" symbol.
2) One dimensional dynamic arrays are allowed.
3) Declaration sequence in global scope can contain ImportedProcedure for declaring procedures and functions from external libraries.

3. Compiler implementation restrictions

1) -2147483648 number not allowed in source text. Use -2147483647 -1 instead of it.
2) Maximal length of identifier is 31 symbols.
3) Maximal length of constant string in source text is 511 symbols.
4) Compiler calculates constant expression only for INTEGER and BOOLEAN types in compile time.

4. Basic types

1) BOOLEAN, the truth values TRUE and FALSE, 1 byte
2) CHAR, 1 byte
3) INTEGER, the integers between -2^31 .. +2^31-1, 4 bytes
4) REAL, real numbers (IEEE standard, 32 bits)
5) LONGREAL, long real numbers (IEEE standard, 64 bits)
6) SET, the sets of integers between 0 and 31, 4 bytes

5. Predefined procedures and functions

Name	Argument type	Result type	Function
ASSERT(b)	BOOLEAN		Abort if ~b
CHR(x)	INTEGER	CHAR	Character with ordinal number x
LEN(v)	v: array	INTEGER	The length of v

Copyright Exaprog.com, 2011

ORD(x)	CHAR	INTEGER	Ordinal number of x

6. The module Memory

NEW(v)	v: pointer to record or pointer to array		Allocate v^.
NEW(v, len)	v: pointer to dynamic array len: INTEGER		Allocate v^ and LEN(v^) = len

7. The module SYSTEM

Name	Argument type	Result type	Function
ADR(v)	any	INTEGER	Address of variable v
MOVE(destination, source, size)	INTEGER, INTEGER, INTEGER		Copies bytes from source address to destination address. It copies size number bytes.

8. Garbage collector

Garbage collector begins to work when NEW is called and there is no free memory at the moment. For garbage collecting it is implemented Mark and Sweep algorithm. Garbage collector doesn't support multi-threading applications.

9. Symbolic debugger

If program aborted then dump file created. It is saved in error.log file in current directory. It contains procedure or module names and list of variables in particular procedure or module. It also contains form of variable (ARRAY, BOOLEAN, CHAR, INTEGER, LONGREAL, POINTER, REAL, RECORD, SET) and value of variable if form is BOOLEAN, CHAR or INTEGER.

10. Compiler checks

Compiler checks indexes of arrays, array sizes in array assignment, record extensions in record assignment and record guards. If all necessary information are available in compile time then compiler checks it in compile time, otherwise it generates code for making checks at run-time in application. If check is false then program is aborted.

Copyright Exaprog.com, 2011

11. Command line arguments

`compiler.exe sourcefile`
Current directory should contains *.sym files for modules which are imported in current `sourcefile`

`compiler.exe -h`
Prints usage information.

`compiler.exe -v`
Prints version information.

`linker.exe projectfile binaryfile`
`projectfile` is file which contains “console” or “window” in the first line for building console or gui application. Following lines in project file should contain modules names in order, which they are initialized.
`binaryfile` is file name for builded application.
Current directory should contains *.sym and *.obj files for modules which are described in `projectfile`.

`linker.exe -h`
Prints usage information.

`linker.exe -v`
Prints version information.

12. Oberon-07 Formal Grammar

```
letter  =  "_" | "A" | "B" | … | "Z" | "a" | "b" … | "z".
digit  =   "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
hexdigit  =  digit | "A" | "B" | "C" | "D" | "E" | "F".
ident  =  letter {letter | digit}.
integer  =  digit {digit} | digit {hexDigit} "H".
real  =  digit {digit} "." {digit} [ScaleFactor].
ScaleFactor  =  ("E" | "D") ["+" | "-"] digit {digit}.
number  =  integer | real.
charConst  =  " ' " character " ' " | digit {hexDigit} "X".
string  =  ' " ' {character} ' " ' | " ' " {character} " ' ".
qualident  =  ident ["." ident].
identdef = ident ["*"].
ConstantDeclaration  =  identdef "=" ConstExpression.
ConstExpression  =  expression.
TypeDeclaration  =  identdef "=" StrucType.
StrucType  =  ArrayType | RecordType | PointerType | ProcedureType.
type  =  qualident | StrucType.
ArrayType  =  "ARRAY" [length {"," length}] "OF" type.
length  =  ConstExpression.
RecordType  =  "RECORD" ["(" BaseType ")"] [FieldListSequence] "END".
```

Copyright Exaprog.com, 2011

```
BaseType  =  qualident.
FieldListSequence  =  FieldList {";" FieldList}.
FieldList  =  IdentList ":" type.
IdentList  =  identdef {"," identdef}.
PointerType  =  "POINTER" "TO" type.
ProcedureType  =  "PROCEDURE" [FormalParameters].
VariableDeclaration  =  IdentList ":" type.
designator  =  qualident {selector} .
selector  =  "." ident | "[" ExpList "]" | "^" |  "(" qualident ")".
ExpList  =  expression {"," expression}.
factor  =  number | CharConst | string | "NIL" | "TRUE" | "FALSE" |
    set | designator [ActualParameters] | "(" expression ")" | "~"
factor.
ActualParameters  =  "(" [ExpList] ")" .
term  =  factor {MulOperator factor}.
MulOperator  =  "*" | "/" | "DIV" | "MOD" | "&".
SimpleExpression  =  ["+" | "-"] term {AddOperator term}.
AddOperator  =  "+" | "-" | "OR".
expression  =  SimpleExpression [relation SimpleExpression].
relation  =  "=" | "#" | "<" | "<=" | ">" | ">=" | "IN" | "IS".
set  =  "{" [element {"," element}] "}".
element  =  expression [".." expression].
statement  =  [assignment | ProcedureCall | IfStatement | CaseStatement |
    WhileStatement | RepeatStatement | ForStatement].
assignment  =  designator ":=" expression.
ProcedureCall  =  designator [ActualParameters].
StatementSequence  =  statement {";" statement}.
IfStatement  =  "IF" expression "THEN" StatementSequence
    {"ELSIF" expression "THEN" StatementSequence}
    ["ELSE" StatementSequence] "END".
CaseStatement  =  "CASE" expression "OF" case {"|" case} "END".
Case  =  CaseLabelList ":"  StatementSequence.
CaseLabelList  = LabelRange {"," LabelRange}.
LabelRange  =  label [".." label].
label  =  integer | ident.
WhileStatement  =  "WHILE" expression "DO" StatementSequence
     {"ELSIF" expression "DO" StatementSequence} "END".
RepeatStatement  =  "REPEAT" StatementSequence "UNTIL" expression.
ForStatement  =  "FOR" ident ":=" expression "TO" expression ["BY"
     ConstExpression] "DO" StatementSequence "END".
ProcedureFlags = ["[" string "," string "," integer "]"].
ImportedProcedure = "PROCEDURE" ProcedureFlags identdef[FormalParameters]
ProcedureDeclaration  =  ProcedureHeading ";" ProcedureBody ident.
ProcedureHeading  =  "PROCEDURE" identdef [FormalParameters].
ProcedureBody  =  DeclarationSequence ["BEGIN" StatementSequence]
    ["RETURN" expression] "END".
DeclarationSequence  =  ["CONST" {ConstDeclaration ";"}]
    ["TYPE" {TypeDeclaration ";"}]
    ["VAR" {VariableDeclaration ";"}]
    {(ProcedureDeclaration | ImportedProcedure) ";"}.
FormalParameters  =  "(" [FPSection {";" FPSection}] ")" [":" qualident].
FPSection  =  ["CONST" | "VAR"] ident {"," ident} ":" FormalType.
FormalType  =  ["ARRAY" "OF"] qualident.
module  =  "MODULE" ident ";" [ImportList] DeclarationSequence
    ["BEGIN" StatementSequence] "END" ident "." .
ImportList  =  "IMPORT" import {"," import} ";".
```

Copyright Exaprog.com, 2011

```
import  =  ident [":=" ident].
```

13. Literature

1. “The Programming Language Oberon Revision” 1.11.2008 Niklaus Wirth

Copyright Exaprog.com, 2011