

Компилятор МультиОберон руководство пользователя

1. Введение.

Copyright © 2019-2021, by [Dmitry Dagaev](#)

Версия 1.2 22-Jul-2021

Отс означает ккомпилятор МультиОберона.

МультиОберон это компилятор языка Оберон с 3 различными бэкендами:

- Генератором нативного кода x86 для системы BlackBox (1.7 с частичной поддержкой 1.6);
- Транслятором Ofront в язык C;
- Генератором кода LLVM.

МультиОберон это кроссплатформенный компилятор с поддержкой:

- Windows X86;
- Windows X64;
- Linux X86;
- Linux X64;
- Raspberry Pi OS ArmV71;
- Ubuntu Linux Aarch64 (64-bit Raspberry Pi).

МультиОберон это масштабируемая технология на основе систем ограничений с начальной точкой в виде синтаксиса Компонентного Паскаля. МультиОберон предназначен для работы из среды BlackBox и из командной строки.

Лицензия GNU Lesser General Public License v3.0.

Адрес загрузки <https://github.com/dvdagaev/Mob>

2. Структура.

МультиОберон состоит из исполняемых файлов приложений и списка программных подсистем. Каталоги бинарных исполняемых файлов соответствуют правилу Bin[os][arch].

arch\os	windows	unix (linux, ...)
X86	Binwe	Binue
X64	Binwr	Binur
ArmV71	--	Binu4
Aarch64	--	Binu8

Бинарные приложения состоят из компиляторов и оболочек (shells). Компиляторы МультиОберона включают в себя полную поддержку функций компиляции и следуют правилу om[backend]c. Оболочки МультиОберона поддерживают только динамическую загрузку и исполнение, файлы соответствуют правилу om[backend]sh.

	BlackBox	Ofront	LLVM
compiler	ombc	omfc	omlc
shell	ombsh	omfsh	omlsh

Программные подсистемы структурируются по правилам подсистем BlackBox. Бэкенд следует правилу Om[backend]. Третья буква означает следующие варианты:

- Омс это компилятор и консоль МультиОберона;
- Омб является реализацией компилятора для бэкенда BlackBox. Основан на Ominc DevCompiler. Линкер ОмбLinker основан на кроссплатформенном решении И.А.Дегтяренко (Trurl);
- Омф является реализацией компилятора для бэкенда Ofront. Основан на Josef Templ's Ofront;
- Омл является реализацией компилятора для бэкенда LLVM. Использует библиотеку, подготовленную из LLVM 5.0.

Каждая подсистема имеет каталог Mod для исходников, Доси для документов, каталоги с кодовыми и символьными файлами.

Каталоги кодовых файлов следуют правилу C[backend][os][arch].

arch\os	windows	Unix (Linux, ...)
X86 BlackBox portable	Sym	Sym
X86 Omb-specific	Cbwe	Cbue
X86 Ofront	Cfwe	Cfue
X64 Ofront	Cfwr	Cfur
X86 LLVM	Clwe	Clue
X64 LLVM	Clwr	Clur
ArmV71 Ofront	--	Cfu4
Aarch64 Ofront	--	Cfu8
ArmV71 LLVM	--	Clu4
Aarch64 LLVM	--	Clu8

Символьные файлы и файлы использования расположены в символьных каталогах.
Каталоги символьных файлов следуют правилу S[backend][os][arch].

arch\os	windows	Unix (Linux, ...)
X86 BlackBox portable	Sym	Sym
X86 Omb-specific	Sbwe	Sbue
X86 OFront	Sfwe	Sfue
X64 OFront	Sfwr	Sfur
X86 LLVM	Slwe	Slue
X64 LLVM	Slwr	Slur
ArmV71 OFront	--	Sfu4
Aarch64 OFront	--	Sfu8
ArmV71 LLVM	--	Slu4
Aarch64 LLVM	--	Slu8

3. Инсталляция.

Краткие инструкции описаны в Bin[os][arch]/INSTALL.txt.

Данный цвет используется для Windows далее и везде.

Данный цвет используется для Unix далее и везде.

3.1 Посредством .msi инсталлятора Windows

Установить MultiOberon_1_2_0_x86.msi для 32-bit или

установить MultiOberon_1_2_0_x64.msi для 64-bit.

Создать рабочий каталог:

```
>mkdir MyDir
```

```
>cd MyDir
```

Установить рабочий каталог ниже.

3.2 Из Github архива Mob-master.zip под Windows

Распаковать multioberon архив в рабочий каталог

```
>unzip Mob-master.zip
```

```
>cd Mob-master
```

Установить рабочий каталог ниже.

3.3 Установка рабочего каталога под Windows

Команда установка рабочего каталога

```
>omc[arch]_install ?/clang? ?BlackBox path?
```

Используйте /clang для работы с clang компилятором/линкером. По умолчанию используется gcc.

```
>omce_install /clang
```

Если BlackBox (<https://blackboxframework.org/>) требуется, установить с путем BlackBox:

```
>omce_install "c:\Program Files (x86)\BlackBox Component Builder 1.7.2"
```

Если BlackBox не требуется, установить без пути BlackBox:

```
>omce_install
```

Установить без BlackBox для 64-bit:

```
>omcr_install
```

3.4 Посредством .deb пакета Unix

Установить multioberon пакет соответствующим:

```
>sudo apt install ./multioberon_i386_1.2.0.deb (* X86 *)
```

```
>sudo apt install ./multioberon_amd64_1.2.0.deb (* X64 *)
```

```
>sudo apt install ./multioberon_armhf_1.2.0.deb (* ArmV71 *)
```

```
>sudo apt install ./multioberon_arm64_1.2.0.deb (* Aarch64 *)
```

Установить рабочий каталог:

```
>mkdir MyDir
```

```
>cd MyDir
```

Для ArmV71

```
>/bin/sh /usr/local/bin/multioberon/omc4_install
```

Для Aarch64

```
>/bin/sh /usr/local/bin/multioberon/omc8_install
```

Если нужен BlackBox, инсталлировать с путем BlackBox:

```
>/bin/sh /usr/local/bin/multioberon/omce_install <path-to-blackbox>; (* ${HOME}/bb *)
```

Если не нужен BlackBox, инсталлировать без параметров:

```
>/bin/sh /usr/local/bin/multioberon/omce_install
```

```
>/bin/sh /usr/local/bin/multioberon/omcr_install
```

Прописать путь PATH= /usr/local/bin/multioberon/Binu[arch] при необходимости.

3.5 Из Github архива Mob-master.zip под Unix

Для Oml - glibc 2.15 or STT_GNU_IFUNC поддержка требуется, старые версии не работают!
Если нужен BlackBox, загрузить bbcb-1.7.2~b1.154.tar.gz и установить 1.7.2 из адреса <https://blackbox.oberon.org/download>

```
>mkdir bb; cd bb; tar xvzf bbcb-1.7.2~b1.154.tar.gz
```

Распаковать multioberon архив в рабочий каталог

```
>rm -fr Mob-master; (* if old *)  
>unzip Mob-master.zip;  
>cd Mob-master
```

Если нужен BlackBox, установить с путем BlackBox:

```
>/bin/sh omc_install <path-to-blackbox>; (* ${HOME}/bb *)
```

Если не нужен BlackBox, установить без параметров:

```
>/bin/sh omc_install
```

Файлы Bin*.zip могут быть удалены после успешной инсталляции.

3.6 Важные замечания

Ожидается, что gcc или clang уже установлен и доступен из командной строки. Иначе, сборка из Ofront и линковка из LLVM не будут работать корректно. Начальная конфигурация устанавливается в Omf.cfg (Oml.cfg) в каталогах Bin[os][arch]. Для примера в Omf.cfg:

```
gcc_opt=-O2 -fshort-wchar  
gcc_lnkopt=-O2 -lm -ldl  
cc=gcc
```

Это означает вызов “gcc -O2 -fshort-wchar” для компиляции и “gcc -O2 -lm -ldl” для линковки. Конфигурация в Oml.cfg аналогична, но требуемое приложение llc уже включено в МультиОберон.

```
llc_opt=-O0  
gcc_lnkopt=-O0 -lm -ldl  
lnk=gcc
```

Для компиляции “llc -O0” вызывается и “gcc -O0 -lm -ldl” вызывается для линковки.

Установки по умолчанию для Windows: cc=clang lnk=clang. Это значит, что clang должен быть предустановлен.

Установки по умолчанию для Unix: cc=gcc lnk=gcc. Это значит, что gcc должен быть предустановлен.

Эти файлы могут конфигурироваться пользователем, но нужно быть внимательным, т.к. каждая инсталляция переписывает их снова.

В среде BlackBox документы Strings используются вместо .cfg файлов:

- Omb/Rsrc/Strings.odc вместо Binwe/Omb.cfg;
- Omf/Rsrc/Strings.odc вместо Binwe/Omf.cfg;
- Oml/Rsrc/Strings.odc вместо Binwe/Oml.cfg

Опции в строковых документах разделяются табуляторами:

```
gcc_opt      -O2  
gcc_lnkopt   -O2  
lnk          clang  
clang_opt    -O2  
clang_lnkopt -O2 -luser32
```

При использовании BlackBox, LLVMT.dll и LLVMT.so должны быть доступны из BlackBox. Например, можно скопировать их в каталог BlackBox.

3 Hello, World из командной строки

Простейший путь запуска модуля OmtestHelloWorld – это выполнение ex[ecute] команды: Bin[os][arch]/om[backend]c ex OmtestHelloWorld. В частности, для BlackBox бэкенда должна быть следующая команда для Windows, x86 Unix (для удобства следует прописать пути, в Windows это делает инсталлятор):

```
>?Pathname?\Binwe\ombc ex OmtestHelloWorld
>?Pathname?\Binue\ombc ex OmtestHelloWorld
Hello, World
```

Для 32-битного бэкенда Ofront используется следующая команда:

```
>?Pathname?\Binwe\omfc ex OmtestHelloWorld
>?Pathname?\Binue\omfc ex OmtestHelloWorld (* X86 *)
>?Pathname?\Binu4\omfc ex OmtestHelloWorld (* ArmV71 *)
Hello, World
```

Для 64-битного бэкенда LLVM используется следующая команда:

```
>?Pathname?\Binwr\omlc ex OmtestHelloWorld
>?Pathname?\Binur\omlc ex OmtestHelloWorld (* X64 *)
>?Pathname?\Binu8\omlc ex OmtestHelloWorld (* Aarch64 *)
Hello, World
```

Далее предполагается, что установлены переменные путей к используемым каталогам Bin*.

Команда execute включает в себя команды compile, build и run. Во-первых, OmtestHelloWorld.mob компилируется в Omtest/Code/HelloWorld.ocf.

```
>ombc co OmtestHelloWorld (* X86 *)
omb:compiling ...HelloWorld.mob >.../Omtest/Code/HelloWorld.ocf code=52
glob=0
```

Расширение .mob означает текстовый файл МультиОберона. Формат файла BlackBox .odc использует опцию -odc. OmtestHelloWorld.odc компилируется в Omtest/Code/HelloWorld.ocf.

```
>ombc co -odc OmtestHelloWorld (* X86 *)
omb:compiling ...HelloWorld.mob >.../Omtest/Code/HelloWorld.ocf code=52
glob=0
```

Компиляция 32-ным Ofront выполняется аналогично, но .c и .h файлы создаются в Cfwe[Cfue] каталоге:

```
>?Pathname?\Binwe\omfc co OmtestHelloWorld
>?Pathname?\Binue\omfc co OmtestHelloWorld
omf:compiling ...HelloWorld.mob >.../Omtest/Cfwe/OmtestHelloWorld
.c=1898 .h=357
```

Компиляция 64-ным LLVM выполняется аналогично, но .ll и .bc файлы создаются в Clwr[Clur] каталоге:

```
>?Pathname?\Binwr\omlc co OmtestHelloWorld
>?Pathname?\Binur\omlc co OmtestHelloWorld
oml:compiling ...HelloWorld.mob >.../Omtest/Clwr/OmtestHelloWorld
.ll=8616 .bc=3172
```

Сборка не требуется для Omb, но Omf нужна внешняя программа для получения объектного файла.

```
>?Pathname?\Binwe\omfc build OmtestHelloWorld
>?Pathname?\Binue\omfc build OmtestHelloWorld
===== building OmtestHelloWorld ... done
```

Аналогично, Oml требует утилиту llc из llvm для создания объектного файла. Однако, llc для LLVM-5.0 включен в дистрибуцию МультиОберона.

```
>?Pathname?\Binwe\omfc build OmtestHelloWorld
>?Pathname?\Binue\omfc build OmtestHelloWorld
===== building OmtestHelloWorld ... done
```

Далее мы можем динамически загружать и выполнять модуль OmtestHelloWorld.

```
>?Pathname?\Binwe\ombc run OmtestHelloWorld
>?Pathname?\Binue\ombc run OmtestHelloWorld
Hello, World
```

32-битный Ofront использует нижеприведенные команды для загрузки и выполнения модуля OmtestHelloWorld.

```
>?Pathname?\Binwe\omfc run OmtestHelloWorld
>?Pathname?\Binue\omfc run OmtestHelloWorld
Hello, World
```

64-битный LLVM использует нижеприведенные команды для загрузки и выполнения модуля OmtestHelloWorld.

```
>?Pathname?\Binwr\omlc run OmtestHelloWorld
>?Pathname?\Binur\omlc run OmtestHelloWorld
Hello, World
```

Исполняемый файл может быть получен из ombc. Линкуйте OmtestHelloWorld.

```
>ombc link -r OmtestHelloWorld (* X86 *)
```

32-битным Ofront исполняемый файл может быть получен из omfc. Линкуйте OmtestHelloWorld.

```
>?Pathname?\Binwe\omfc link -r OmtestHelloWorld
>?Pathname?\Binue\omfc link -r OmtestHelloWorld
```

64-битным LLVM исполняемый файл может быть получен из omlc. Линкуйте OmtestHelloWorld.

```
>?Pathname?\Binwe\omlc link -r OmtestHelloWorld
>?Pathname?\Binue\omlc link -r OmtestHelloWorld
```

Наконец, мы можем запустить OmtestHelloWorld программы BlackBox.

```
>cd Omtest/Code (* X86 *)
>OmtestHelloWorld
Hello, World
```

Также можно запустить для 32-ного Ofront.

```
>Omtest\Cfwe\OmtestHelloWorld
>Omtest\Cfue\OmtestHelloWorld
>Omtest\Cfu4\OmtestHelloWorld
Hello, World
```

Также можно запустить для 64-ного LLVM.

```
>Omtest\Clwr\OmtestHelloWorld
>Omtest\Clur\OmtestHelloWorld
>Omtest\Clu8\OmtestHelloWorld
Hello, World
```

Модули 64-ного Ofront, 32-ного LLVM, ArmV71, Aarch64 могут компилироваться аналогичным образом.

4 Omb – запуск из среды Black Box

5.1 Инсталляция

Предварительно.

Omb не использует никаких сервисов, кроме BlackBox.

Запуск: StartupBlackBox

Открыть Omb/Docu/Quick-Start.odc и примеры из Omtest/Docu/Quick-Start.odc

5.2 Компилирование Omb

Компилируйте модули из BlackBox:

1.3. Compile the following portable modules:

● DevCompiler.CompileThis OmcCfgfile OmcTarget OmcCRuntime OmcHooks OmcDialog OmcOPM OmcOPT OmcOPU OmcOPB OmcOPS OmcOPP OmcDump OmcTester OmcParams OmcCommandParams OmcOdcSource OmcTxtSource OmcRuntimeStd OmcDialogStd OmcDialogConsole OmcCompiler OmbOPE OmbOPH OmbOPL486 OmbOPC486 OmbOPV486 OmbInlink OmbLnkBase OmbLnkLoad OmbLnkWritePe OmbLnkWriteElf OmbLnkWriteElfStatic OmbLinkPortableProcessor OmbParams OmbBackEnd OmbCompiler OmbLinker

Все команды BlackBox далее оформляются в следующем виде:

```
^Q DevCompiler.CompileThis OmcCfgfile OmcTarget OmcCRuntime OmcHooks  
OmcDialog OmcOPM OmcOPT OmcOPU OmcOPB OmcOPS OmcOPP OmcDump OmcTester  
OmcParams OmcCommandParams OmcOdcSource OmcTxtSource OmcRuntimeStd  
OmcDialogStd OmcDialogConsole OmcCompiler OmbOPE OmbOPH OmbOPL486  
OmbOPC486 OmbOPV486 OmbInlink OmbLnkBase OmbLnkLoad OmbLnkWritePe  
OmbLnkWriteElf OmbLnkWriteElfStatic OmbLinkPortableProcessor OmbParams  
OmbBackEnd OmbCompiler OmbLinker
```

5.3 Компилирование примеров из Omtest

Компилируйте модули:

```
^Q OmbCompiler.CompileThis OmtestHelloWorld OmtestFormats OmtestDateTime  
OmtestMkTraps OmtestHeap
```

5.4 Запуск примеров из Omtest

5.4.1 Простейший пример Hello, World

```
^Q OmtestHelloWorld.MAIN
```

5.4.2 Форматирование строки, целых и действительных чисел

```
^Q OmtestFormats.MAIN
```

5.4.3 Дата, время и задержки

```
^Q OmtestDateTime.MAIN
```

5.4.4 Обработка трапов рантаймом

Простой Assert

```
^Q "OmtestMkTraps.RunOpt('a')"
```

Простой Halt

```
^Q "OmtestMkTraps.RunOpt('h')"
```

Деление на ноль

```
^Q "OmtestMkTraps.RunOpt('z')"
```

Разыменованное нулевого указателя

```
^Q "OmtestMkTraps.RunOpt('p')"
```

5.4.5 Динамическая память и сборка мусора

! программа виснет на Omb Unix т.к. Kernel для BlackBox для Unix неоптимально работает с памятью

```
^Q OmbTestHeap.MAIN
```

5.5 Создание исполняемых файлов

Линковка исполняемых файлов для запуска из консоли. Опция -r добавляет рекурсивно все модули импортирования. Иначе явное добавление Kernel\$+ Log Math Strings OStrings OLog HostConLog Runner должно быть выполнено.

```
^Q OmbLinker.LinkExe -r -o "OmbTest/Code/OmbTestHelloWorld"
OmbTestHelloWorld
^Q OmbLinker.LinkExe -r -o "OmbTest/Code/OmbTestFormats" OmbTestFormats
^Q OmbLinker.LinkExe -r -o "OmbTest/Code/OmbTestDateTime.exe"
OmbTestDateTime
^Q OmbLinker.LinkExe -r -o "OmbTest/Code/OmbTestMkTraps.exe" OmbTestMkTraps
^Q OmbLinker.LinkExe -r -o "OmbTest/Code/OmbTestHeap.exe" OmbTestHeap
```

Скомпилированные файлы запускаются из консоли:

```
>cd OmbTest/Code
>OmbTestHelloWorld
Hello, World
```

5.6 [Продвинутое] Само-компиляция компилятора Omb

Компилятор Omb (OmbCoSh) представляет собой консольное приложение exe с полным набором функций компиляции. См самокомпиляцию портативного компилятора.

```
^Q OmbCompiler.CompileThis OmbCfgfile OmbTarget OmbCRuntime OmbHooks
OmbDialog OmbOPM OmbOPT OmbOPU OmbOPB OmbOPS OmbOPP OmbDump OmbTester
OmbParams OmbCommandParams OmbOdcSource OmbTxtSource OmbRuntimeStd
OmbDialogStd OmbDialogConsole OmbTimesDialog OmbCompiler OmbConsole
OmbDiscomp OmbOPE OmbOPH OmbOPL486 OmbOPC486 OmbOPV486 OmbInlink
OmbLnkBase OmbLnkLoad OmbLnkWritePe OmbLnkWriteElf OmbLnkWriteElfStatic
OmbLinkPortableProcessor OmbParams OmbBackEnd OmbCompiler
OmbLoaderRoutines OmbOcfLoader OmbLinker OmbCoSh
```

Для Windows линковка консольного компилятора

```
^Q OmbLinker.LinkExe -r -tl 2 -o "Binwe/ombc" $+Kernel Log Math Strings
OStrings OLog HostConLog Runner Files HostFiles OmbCfgfile Dates Times
HostTimes Dialog Stores Sequencers Models Services Fonts Meta Converters
Ports Views Controllers Properties Mechanisms Containers Printers
Printing OmbTimesDialog Documents TextModels TextRulers TextSetters
TextViews OmbTarget OmbCRuntime OmbDialog OmbHooks OmbOPM OmbOPT OmbOPB
OmbOPU OmbOPS OmbOPP OmbTester OmbParams OmbOdcSource OmbDialogConsole
OmbRuntimeStd OmbConsole OmbDiscomp OmbLoaderRoutines OmbOcfLoader OmbOPE
OmbOPH OmbOPL486 OmbOPC486 OmbOPV486 OmbBackEnd OmbInlink OmbLnkBase
OmbLnkLoad OmbLnkWritePe OmbLnkWriteElf OmbLnkWriteElfStatic
OmbLinkPortableProcessor OmbCoSh
```

Для Linux линковка консольного компилятора

```
^Q OmbLinker.LinkExe -o "Binue/ombc" $+Kernel Log Math Strings OStrings
OLog HostConLog Runner Testing Files HostFiles OmbCfgfile Dates Times
HostTimes Dialog Stores Sequencers Models Services Fonts Meta Converters
Ports Views Controllers Properties Mechanisms Containers Printers
Printing OmbTimesDialog Documents TextModels TextRulers TextSetters
TextViews OmbTarget OmbCRuntime OmbDialog OmbHooks OmbOPM OmbOPT OmbOPB
```

```
OmcOPU OmcOPS OmcOPP OmcTester OmcParams OmcOdcSource OmcDialogConsole  
OmcRuntimeStd OmcConsole OmcDiscomp OmbOPE OmbOPH OmbOPL486 OmbOPC486  
OmbOPV486 OmbBackEnd OmcLoaderRoutines OmcOcfLoader OmbInlink OmbLnkBase  
OmbLnkLoad OmbLnkWritePe OmbLnkWriteElf OmbLnkWriteElfStatic  
OmbLinkPortableProcessor OmbCoSh
```

Опция `-r` не используется здесь, т.к. множество модулей, например `TextModels`, не импортируются прямо, они должны быть явно указаны линкеру.

5.7 [Продвинутое] Компиляция минимальной оболочки `ombsh`

Минимальная `Omb` оболочка (`OmbShell`) – это консольное приложение с динамической загрузкой и поддержкой выполнения.

Для Windows компиляция `OmbShell`

```
^Q OmbCompiler.CompileThis OmcLoaderRoutines OmcOcfLoader  
OmcObjLoader_Coff OmcShell OmbShell
```

Для Windows линковка `OmbShell`

```
^Q OmbLinker.LinkExe -r -tl 2 -o "Binwe/ombsh" $+Kernel Log Math Strings  
OStrings OLog HostConLog Runner Files HostFiles OmcLoaderRoutines  
OmcOcfLoader OmcObjLoader OmcShell OmbShell
```

Для Linux компиляция `OmbShell`

```
^Q OmbCompiler.CompileThis OmcLoaderRoutines OmcOcfLoader  
OmcObjLoader_Elf OmcShell OmbShell
```

Для Linux линковка `OmbShell`

```
^Q OmbLinker.LinkExe -r -tl 2 -o "Binue/ombsh" $+Kernel Log Math Strings  
OStrings OLog HostConLog Runner Files HostFiles OmcLoaderRoutines  
OmcOcfLoader OmcObjLoader OmcShell OmbShell
```

5.8 Выгрузка компилятора `Omb`

```
^Q DevDebug.UnloadThis OmbLinker OmbCompiler OmbBackEnd OmbParams  
OmbLinkPortableProcessor OmbLnkWriteElfStatic OmbLnkWriteElf  
OmbLnkWritePe OmbLnkLoad OmbLnkBase OmbLinkWinProcessor OmbInlink  
OmcLoader OmcCompiler OmcCommandParams OmbOPV486 OmbOPC486 OmbOPL486  
OmbOPH OmbOPE OmcTimesDialog OmcDialogStd OmcRuntimeStd OmcOdcSource  
OmcParams OmcTester OmcDump OmcOPP OmcOPS OmcOPU OmcOPB OmcOPT OmcOPM  
OmcDialog OmcHooks OmcCRuntime OmcTarget OmcCfgfile Runner Testing
```

5 Оmb из командной строки.

6.1 Инсталляция

Предварительно.

Omb не использует других сервисов. Выполняйте все команды ниже из корневого каталога Mob-master.

6.2 Компиляция примеров

Скрипт для компиляции и линковки всех примеров.

```
B\bwe_tomake.bat  
B/bue_tomake.sh      (* x86 *)
```

Удаляет все приложения с примерами

```
B\bwe_toclean  
B/bue_toclean.sh
```

Ниже приведен набор команд компиляции всех примеров

```
?Pathname?\Binwe\ombc co -odc OmtestHelloWorld OmtestFormats  
OmtestDateTime OmtestMkTraps OmtestHeap  
?Pathname?\Binwe\ombc link -r OmtestHelloWorld  
?Pathname?\Binwe\ombc link -r OmtestFormats  
?Pathname?\Binwe\ombc link -r OmtestDateTime  
?Pathname?\Binwe\ombc link -r OmtestMkTraps  
?Pathname?\Binwe\ombc link -r OmtestHeap  
?Pathname?\Binue\ombc co -odc OmtestHelloWorld OmtestFormats  
OmtestDateTime OmtestMkTraps OmtestHeap  
mkdir -p Omtest/Cbue  
?Pathname?\Binue\ombc link -r OmtestHelloWorld  
?Pathname?\Binue\ombc link -r OmtestFormats  
?Pathname?\Binue\ombc link -r OmtestDateTime  
?Pathname?\Binue\ombc link -r OmtestMkTraps  
?Pathname?\Binue\ombc link -r OmtestHeap
```

Первая команда списка компилирует все файлы примеров. Линковка создает требуемый набор приложений. Если мы опускаем -г опцию рекурсивной работы, команда линковщика должна быть аналогично следующей:

```
?Pathname?\Binwe\ombsh li $+Kernel Log Math Strings OStrings OLog  
HostConLog Runner OmtestHelloWorld  
?Pathname?\Binue\ombsh li $+Kernel Log Math Strings OStrings OLog  
HostConLog Runner OmtestHelloWorld
```

6.3 Выполнение примеров

6.3.1 Простейший Hello, World

```
>Omtest\Cbwe\OmtestHelloWorld  
>Omtest/Cbue/OmtestHelloWorld  
Hello, World
```

6.3.2 Форматирование строки, целых и действительных чисел

```
>Omtest\Cbwe\OmtestFormats  
>Omtest/Cbue/OmtestFormats
```

6.3.3 Дата, время и задержки

```
>Omtest\Cbwe\OmtestDateTime  
>Omtest/Cbue/OmtestDateTime
```

6.3.4 Обработка трапов рантаймом

Простой Assert

```
>Omtest\Cbwe\OmtestMkTraps -trap a  
>Omtest/Cbue/OmtestKmTraps -trap a
```

Простой Halt

```
>Omtest\Cbwe\OmtestMkTraps -trap h  
>Omtest/Cbue/OmtestKmTraps -trap h
```

Деление на ноль

```
>Omtest\Cbwe\OmtestMkTraps -trap z  
>Omtest/Cbue/OmtestKmTraps -trap z
```

Разыменование нулевого указателя

```
>Omtest\Cbwe\OmtestMkTraps -trap p  
>Omtest/Cbue/OmtestKmTraps -trap p
```

6.3.5 Динамическая память и сборка мусора

```
>Omtest\Cbwe\OmtestHeap  
>Omtest/Cbue/OmtestHeap
```

6.4 Само-компиляция

Скрипт для компилирования компилятора ombsc представлен ниже:

```
B\bwe_compiler_tomake.bat  
B/bue_compiler_tomake.sh
```

Скрипт для компилирования минимальной оболочки ombsh представлен ниже:

```
B\bwe_sh_tomake.bat  
B/bue_sh_tomake.sh
```

6 Omf Ofront – запуск из среды Black Box

Omf бэкенда Ofront создает программный код (*.c) и заголовочные файлы (*.h), которые должны быть скомпилированы в объектные файлы (*.o). Т.к. объектные файлы не могут динамически загружаться в среду BlackBox, функциональность Omf Ofront ограничена созданием файлов. Ни динамическая загрузка, ни выполнение не работают в среде BlackBox.

7.1 Инсталляция

Предварительно.

Omf использует ранее установленный C/C++ компилятор gcc или clang.

Запуск: StartupBlackBox

Открыть Omf/Docu/Quick-Start.odc и примеры Omtest/Docu/Quick-Start.odc

7.2 Компиляция Omf

Компилируйте модули для Omf:

3. Compile the following modules:

```
❶ DevCompiler.CompileThis OmcCfgfile OmcTarget OmcCRuntime OmcHooks OmcDialog OmcOPM  
OmcOPT OmcOPU OmcOPB OmcOPS OmcOPP OmcDump OmcTester OmcParams OmcCommandParams  
OmcOdcSource OmcTxtSource OmcOdcTextReader OmcExtSource OmcRuntimeStd OmcDialogStd  
OmcDialogConsole OmcCompiler OmcTimesDialog OmcConsole OmfOPG OmfOPC OmfOPV OmfParams  
OmfBackEnd OmfCompiler OmfLinker
```

Все команды BlackBox далее оформляются в следующем виде:

```
^Q DevCompiler.CompileThis OmcCfgfile OmcTarget OmcCRuntime OmcHooks  
OmcDialog OmcOPM OmcOPT OmcOPU OmcOPB OmcOPS OmcOPP OmcDump OmcParams  
OmcCommandParams OmcOdcSource OmcTxtSource OmcOdcTextReader OmcExtSource  
OmcRuntimeStd OmcDialogStd OmcDialogConsole OmcCompiler OmcTimesDialog  
OmcConsole OmfOPG OmfOPC OmfOPV OmfParams OmfBackEnd OmfCompiler  
OmfLinker
```

7.3 Компиляция примеров Omtest

Компилируйте модули ниже (‘:’ означает создание main для модуля):

```
^Q OmfCompiler.CompileThis :OmtestHelloWorld :OmtestFormats  
:OmtestDateTime :OmtestMkTraps :OmtestHeap
```

Ожидаемый результат в ~/Omtest/Cfwe/ каталоге (Cfue для Unix): OmtestHelloWorld.c
OmtestFormats.c OmtestDateTime.c OmtestMkTraps.c OmtestHeap.c

Компилируйте модули ниже для 64 бит:

```
^Q OmfCompiler.CompileThis -64 :OmtestHelloWorld :OmtestFormats  
:OmtestDateTime :OmtestMkTraps :OmtestHeap
```

Ожидаемый результат в ~/Omtest/Cfwr/ каталоге (Cfur для Unix): OmtestHelloWorld.c
OmtestFormats.c OmtestDateTime.c OmtestMkTraps.c OmtestHeap.c

Построение объектных файлов требует использовать внешнюю консольную программу типа gcc. Лучше это запускать из консоли. В принципе можно делать это из BlackBox, но это неудобно и на время блокирует систему.

```
^Q OmfLinker.BuildFiles -r OmtestHelloWorld
```

Линковка аналогична:

```
^Q OmfLinker.LinkExe -r OmtestHelloWorld
```

7.4 [Продвинутое] Компиляция минимальной оболочки omfsh

Минимальная Omf оболочка (OmfShell) представляет собой консольное приложение с поддержкой динамической загрузки и выполнения модулей.

Для Windows компиляция OmfShell

```
^Q OmfCompiler.CompileThis OmcLoaderRoutines OmcObjLoader__Coff OmcShell  
:OmfShell  
^Q OmfCompiler.CompileThis -64 OmcLoaderRoutines OmcObjLoader__Coff  
OmcShell :OmfShell
```

Для Linux-specific компиляция omfsh

```
^Q OmfCompiler.CompileThis OmcLoaderRoutines OmcObjLoader__Elf OmcShell  
:OmfShell  
^Q OmfCompiler.CompileThis -64 OmcLoaderRoutines OmcObjLoader__Elf  
OmcShell :OmfShell
```

7.5 Выгрузка компилятора Omf

```
^Q DevDebug.UnloadThis OmfCompiler OmfLinker OmfBackEnd OmfParams OmfOPV  
OmfOPC OmfOPG OmcCoffLoader OmcLoaderRoutines OmcCompiler OmcDialogStd  
OmcRuntimeStd OmcLogStd OmcOdcSource OmcCommandParams OmcParams OmcTester  
OmcDump OmcOPP OmcOPS OmcOPU OmcOPB OmcOPT OmcOPM OmcDialog OmcHooks  
OmcCRuntime OmcTarget Runner
```

7 Omfc Ofront из командной строки.

8.1 Инсталляция

Предварительно.

Omfc использует ранее инсталлированный C/C++ компилятор gcc или clang. Omfc.cfg должен содержать имена внешнего компилятора и опции со значениями (см 3. - Инсталляция). Omfc может использоваться для получения omfc. Выполняйте все команды ниже из корневого каталога Mob-master.

8.2 Компиляция примеров

Скрипт для компиляции и линковки примеров.

```
B\fwc_tomake.bat          (* X86 *)
B/fwc_tomake.sh           (* X86 *)
B\fwr_tomake.bat          (* X64 *)
B/fwr_tomake.sh           (* X64 *)
B/fu4_tomake.sh           (* ArmV7l *)
B/fu8_tomake.sh           (* Aarch64 *)
```

Cleans Удаляет все приложения с примерами

```
fwc_toclean
fwc_toclean.sh
```

Ниже приведен набор команд компиляции всех примеров

```
?Pathname?\Binwe\omfc co -odc :OmtestHelloWorld :OmtestFormats
:OmtestDateTime :OmtestMkTraps :OmtestHeap
?Pathname?\Binwe\omfc build -r OmtestDateTime
?Pathname?\Binwe\omfc build OmtestHelloWorld
?Pathname?\Binwe\omfc build OmtestFormats
?Pathname?\Binwe\omfc build OmtestMkTraps
?Pathname?\Binwe\omfc build OmtestHeap
?Pathname?\Binwe\omfc link -r OmtestDateTime
?Pathname?\Binwe\omfc link -r OmtestHelloWorld
?Pathname?\Binwe\omfc link -r OmtestFormats
?Pathname?\Binwe\omfc link -r OmtestMkTraps
?Pathname?\Binwe\omfc link -r OmtestHeap
?Pathname?\Binue\omfc co -odc :OmtestHelloWorld :OmtestFormats
:OmtestDateTime :OmtestMkTraps :OmtestHeap
?Pathname?\Binue\omfc build -r OmtestDateTime
?Pathname?\Binue\omfc build OmtestHelloWorld
?Pathname?\Binue\omfc build OmtestFormats
?Pathname?\Binue\omfc build OmtestMkTraps
?Pathname?\Binue\omfc build OmtestHeap
?Pathname?\Binue\omfc link -r OmtestDateTime
?Pathname?\Binue\omfc link -r OmtestHelloWorld
?Pathname?\Binue\omfc link -r OmtestFormats
?Pathname?\Binue\omfc link -r OmtestMkTraps
?Pathname?\Binue\omfc link -r OmtestHeap
```

Первая команда списка компилирует все файлы примеров. Линковка создает требуемый набор приложений. Мы используем ':' для модулей с main и -r опцию рекурсивной работы.

8.3 Выполнение примеров

8.3.1 Простейший пример Hello, World

```
>Omtest\Cfwe\OmtestHelloWorld
>Omtest\Cfur\OmtestHelloWorld
>Omtest\Cfu4\OmtestHelloWorld
>Omtest\Cfu8\OmtestHelloWorld
Hello, World
```

8.3.2 Форматирование строки, целых и действительных чисел

```
>Omtest\Cfwe\OmtestFormats
>Omtest\Cfue\OmtestFormats
```

8.3.3 Дата, время и задержки

```
>Omtest\Cfwe\OmtestDateTime
>Omtest\Cfue\OmtestDateTime
```

8.3.4 Обработка трапов рантаймом

Простой Assert

```
>Omtest\Cfwe\OmtestMkTraps -trap a
>Omtest\Cfue\OmtestKmTraps -trap a
```

Простой Halt

```
>Omtest\Cfwe\OmtestMkTraps -trap h
>Omtest\Cfue\OmtestKmTraps -trap h
```

Деление на ноль

```
>Omtest\Cfwe\OmtestMkTraps -trap z
>Omtest\Cfue\OmtestKmTraps -trap z
```

Разыменование нулевого указателя

```
>Omtest\Cfwe\OmtestMkTraps -trap p
>Omtest\Cfue\OmtestKmTraps -trap p
```

8.3.5 Динамическая память и сборка мусора

```
>Omtest\Cfwe\OmtestHeap
>Omtest\Cfue\OmtestHeap
```

8.4 Само и кросс компиляция

Скрипт для само компиляции omfc следующий:

```
B\fwe_compiler_tomake.bat      (* X86 *)
B\fwr_compiler_tomake.bat      (* X64 *)
B/fue_compiler_tomake.sh       (* X86 *)
B/fur_compiler_tomake.sh       (* X64 *)
B/fu4_compiler_tomake.sh       (* ArmV71 *)
B/fu8_compiler_tomake.sh       (* Aarch64 *)
```

Скрипт для компиляции минимальной оболочки omfsh следующий:

```
B\fwe_sh_tomake.bat
B/fue_sh_tomake.sh
```


9. Oml LLVM – запуск из среды Black Box

Oml LLVM бэкэнд создает текстовые (*.ll) и бинарные (*.bc) файлы байт-кода, которые затем компилируются в объектные файлы (*.o). Т.к. объектные файлы не могут быть загружены в среду BlackBox, функциональность Oml LLVM ограничена созданием файлов. Ни динамическая загрузка, ни выполнение не работают в среде BlackBox.

9.1 Инсталляция

Предварительно.

Oml использует LLVM 5.0 в библиотеке LLVMT.dll / LLVMT.so и llvm llc утилиту, включаемую в дистрибуцию МультиОберон.

Запуск: StartupBlackBox

Открыть Oml/Docu/Quick-Start.odc и примеры Omltest/Docu/Quick-Start.odc

9.2 Компиляция Oml

Список модулей Oml компилируется для среды BlackBox в два этапа: Служебные LLVM и Oml компилятор для BlackBox:

1.3 Compile LLVM Services

```
❶ DevCompiler.CompileThis LlvmC LlvmForAArch64 LlvmForAMDGPU LlvmForARM LlvmForBPF  
LlvmForHexagon LlvmForLanai LlvmForMips LlvmForMSP430 LlvmForNVPTX LlvmForPowerPC LlvmForSparc  
LlvmForSystemZ LlvmForX86 LlvmForXCore LlvmNative LlvmRefs  
❷ DevCompiler.CompileThis OmcCfgfile OmcTarget OmcCRuntime OmcHooks OmcDialog OmcOPM  
OmcOPT OmcOPU OmcOPB OmcOPS OmcOPP OmcDump OmcTester OmcParams OmcCommandParams  
OmcOdcSource OmcOdcTextReader OmcExtSource OmcRuntimeStd OmcDialogStd OmcDialogConsole  
OmcCompiler OmcConsole OmlOPG OmlOPL OmlOPF OmlOPC OmlOPV OmlParams OmlBackEnd  
OmlCompiler OmlLinker
```

Все команды BlackBox далее оформляются в следующем виде:

```
^Q DevCompiler.CompileThis LlvmC LlvmForAArch64 LlvmForAMDGPU LlvmForARM  
LlvmForBPF LlvmForHexagon LlvmForLanai LlvmForMips LlvmForMSP430  
LlvmForNVPTX LlvmForPowerPC LlvmForSparc LlvmForSystemZ LlvmForX86  
LlvmForXCore LlvmNative LlvmRefs  
^Q DevCompiler.CompileThis OmcCfgfile OmcTarget OmcCRuntime OmcHooks  
OmcDialog OmcOPM OmcOPT OmcOPU OmcOPB OmcOPS OmcOPP OmcDump OmcTester  
OmcParams OmcCommandParams OmcOdcSource OmcOdcTextReader OmcExtSource  
OmcRuntimeStd OmcDialogStd OmcDialogConsole OmcCompiler OmcConsole OmlOPG  
OmlOPL OmlOPF OmlOPC OmlOPV OmlParams OmlBackEnd OmlCompiler OmlLinker
```

9.3 Компилятор примеров

Компилируйте модули ниже (‘.’ означает создание main для модуля):

```
^Q OmlCompiler.CompileThis :OmltestHelloWorld :OmltestFormats  
:OmltestDateTime :OmltestMkTraps :OmltestHeap
```

Ожидаемый результат в ~/Omltest/Clwe/ каталоге (Clue для Unix): OmltestHelloWorld.ll
OmltestFormats.ll OmltestDateTime.ll OmltestMkTraps.ll OmltestHeap.ll

Компилируйте модули для 64 бит:

```
^Q OmlCompiler.CompileThis -64 :OmltestHelloWorld :OmltestFormats  
:OmltestDateTime :OmltestMkTraps :OmltestHeap
```

Ожидаемый результат в ~/Omltest/Clur/ каталоге (Clur для Unix): OmltestHelloWorld.ll
OmltestFormats.ll OmltestDateTime.ll OmltestMkTraps.ll OmltestHeap.ll

Создание объектников требует вызова внешней консольной программы lsc. Лучше это запускать из консоли. В принципе можно делать это из BlackBox, но это неудобно и на время блокирует систему.

```
^Q OmlLinker.BuildFiles -r OmtestHelloWorld
```

Линковка аналогична:

```
^Q OmlLinker.LinkExe -r OmtestHelloWorld
```

9.4 [Продвинутое] Компиляция минимальной оболочки omlsh

Минимальная Oml оболочка (OmlShell) представляет собой консольное приложение с поддержкой динамической загрузки и выполнения модулей.

Для Windows компиляция omlsh

```
^Q OmlCompiler.CompileThis OmcLoaderRoutines OmcObjLoader__Coff OmcShell  
OmlBcLoader :OmlShell  
^Q OmlCompiler.CompileThis -64 OmcLoaderRoutines OmcObjLoader__Coff  
OmcShell OmlBcLoader :OmlShell
```

Для Linux компиляция omlsh

```
^Q OmlCompiler.CompileThis OmcLoaderRoutines OmcObjLoader__Elf OmcShell  
OmlBcLoader :OmlShell  
^Q OmlCompiler.CompileThis -64 OmcLoaderRoutines OmcObjLoader__Elf  
OmcShell OmlBcLoader :OmlShell
```

9.5 Выгрузка компилятора Oml

```
^Q DevDebug.UnloadThis OmlCompiler OmlLinker OmlBackEnd OmlParams OmlOPV  
OmlOPC OmlOPF OmlOPL OmlOPG OmcCompiler OmcDialogStd OmcRuntimeStd  
OmcOdcSource OmcCommandParams OmcParams OmcTester OmcDump OmcOPP OmcOPS  
OmcOPU OmcOPB OmcOPT OmcOPM OmcDialog OmcHooks OmcCRuntime OmcTarget  
OmcCfgfile Runner OLog HostTimes Times OStrings Testing
```

10. Oml LLVM - из командной строки.

10.1 Инсталляция

Предварительно.

Oml использует LLVM 5.0 в библиотеке LLVMT.dll / LLVMT.so и утилиту llvm llc включаемую в дистрибуцию МультиОберон. Oml.cfg должен содержать имена внешнего компилятора и опции со значениями (см 3. - Инсталляция). Выполняйте все команды ниже из корневого каталога Mob-master.

10.2 Компиляция примеров

Скрипт для компиляции и линковки примеров.

```
B\lwe_tomake.bat      (* X86 *)
B/lue_tomake.sh       (* X86 *)
B\lwr_tomake.bat      (* X64 *)
B/lur_tomake.sh       (* X64 *)
B/lu4_tomake.sh       (* ArmV7l *)
B/lu8_tomake.sh       (* Aarch64 *)
```

Удаляет все приложения с примерами

```
B\lwe_toclean
B/lue_toclean.sh
```

Ниже приведен набор команд компиляции всех примеров

```
?Pathname?\Binwe\omlc co -odc :OmtestHelloWorld :OmtestFormats
:OmtestDateTime :OmtestMkTraps :OmtestHeap
?Pathname?\Binwe\omlc build -r OmtestDateTime
?Pathname?\Binwe\omlc build OmtestHelloWorld
?Pathname?\Binwe\omlc build OmtestFormats
?Pathname?\Binwe\omlc build OmtestMkTraps
?Pathname?\Binwe\omlc build OmtestHeap
?Pathname?\Binwe\omlc link -r OmtestDateTime
?Pathname?\Binwe\omlc link -r OmtestHelloWorld
?Pathname?\Binwe\omlc link -r OmtestFormats
?Pathname?\Binwe\omlc link -r OmtestMkTraps
?Pathname?\Binwe\omlc link -r OmtestHeap
?Pathname?\Binue\omlc co -odc :OmtestHelloWorld :OmtestFormats
:OmtestDateTime :OmtestMkTraps :OmtestHeap
?Pathname?\Binue\omlc build -r OmtestDateTime
?Pathname?\Binue\omlc build OmtestHelloWorld
?Pathname?\Binue\omlc build OmtestFormats
?Pathname?\Binue\omlc build OmtestMkTraps
?Pathname?\Binue\omlc build OmtestHeap
?Pathname?\Binue\omlc link -r OmtestDateTime
?Pathname?\Binue\omlc link -r OmtestHelloWorld
?Pathname?\Binue\omlc link -r OmtestFormats
?Pathname?\Binue\omlc link -r OmtestMkTraps
?Pathname?\Binue\omlc link -r OmtestHeap
```

Первая команда списка компилирует все файлы примеров. Линковка создает требуемый набор приложений. Мы используем ':' для модулей с main и -r опцию рекурсивной работы.

10.3 Выполнение примеров

10.3.1 Простейший пример Hello, World

```
>Omtest\Clwe\OmtestHelloWorld      (* X86 *)
>Omtest/Clur/OmtestHelloWorld      (* X64 *)
>Omtest/Clu4/OmtestHelloWorld      (* ArmV71 *)
>Omtest/Clu8/OmtestHelloWorld      (* Aarch64 *)
Hello, World
```

10.3.2 Форматирование строки, целых и действительных чисел

```
>Omtest\Clwe\OmtestFormats
>Omtest/Clue/OmtestFormats
```

10.3.3 Дата, время и задержки

```
>Omtest\Clwe\OmtestDateTime
>Omtest/Clue/OmtestDateTime
```

10.3.4 Обработка трапов рантаймом

Простой Assert

```
>Omtest\Clwe\OmtestMkTraps -trap a
>Omtest/Clue/OmtestKmTraps -trap a
```

Простой Halt

```
>Omtest\Clwe\OmtestMkTraps -trap h
>Omtest/Clue/OmtestKmTraps -trap h
```

Деление на ноль

```
>Omtest\Clwe\OmtestMkTraps -trap z
>Omtest/Clue/OmtestKmTraps -trap z
```

Разыменование нулевого указателя

```
>Omtest\Clwe\OmtestMkTraps -trap p
>Omtest/Clue/OmtestKmTraps -trap p
```

10.3.5 Динамическая память и сборка мусора

```
>Omtest\Clwe\OmtestHeap
>Omtest/Clue/OmtestHeap
```

10.4 Само-компиляция

Скрипт для само компиляции om!c следующий:

```
B\lwe_compiler_tomake.bat      (* X86 *)
B\lwr_compiler_tomake.bat      (* X64 *)
B\lue_compiler_tomake.sh       (* X86 *)
B\lur_compiler_tomake.sh       (* X64 *)
B\lu4_compiler_tomake.sh       (* ArmV71 *)
B\lu8_compiler_tomake.sh       (* Aarch64 *)
```

Скрипт для компиляции минимальной 64-битной оболочки om!sh следующий:

```
B\lwr_sh_tomake.bat
B\lur_sh_tomake.sh
```

11. Файлы использования и обход дерева импорта

МультОберон добавляет файлы использования *.ouf, которые содержат списки импорта и информацию об ограничениях. Эти файлы расположены в Sym или S[backend][os][arch] каталогах. Бинарный файл использования имеет формат (item, value)* :

- item – целочисленный номер опции;
- value – строковая константа как значение опции.

Следующие опции используются в настоящий момент:

16	sNAME	Имя модуля
50	sIMPNAME	Имя импортируемого модуля
51	sIMALIAS	Псевдоним импортируемого модуля
52	sLIBCODE	Имя библиотеки в модуле с кодом
53	sLIBNOCODE	Имя библиотеки в модуле без кода

Файлы использования позволяют обходить (траверс) все дерево импортируемых модулей.

Пример – команда tr[averse]:

```
>Binwe\ombc trav OmtestHelloWorld
>Binue/ombc trav OmtestHelloWorld
OmtestHelloWorld
  Runner
    SYSTEM
    ?c:\suok5\dsu\System\Slwe\SYSTEM.ouf
    Kernel
      Api
        -Api [libcmt]
      OLog
        OStrings
        -OStrings
      -OLog
    -Kernel
  -Runner
-OmtestHelloWorld
```

Модуль Runner импортируется из OmtestHelloWorld, модули SYSTEM и Kernel импортируются из Runner, модули Api и OLog импортируются из Kernel. Модуль SYSTEM никогда не компилировался, поэтому файл SYSTEM.ouf отсутствует. Модуль Api имеет библиотеку 'libcmt'.

Следует заметить, что импортная информация специфична для бэкенда, она может располагаться в файлах .ocf для BlackBox и .c для Ofront. Для избежания специфического, требующего времени, парсинга при получении списков импорта модулей было решено добавить явно файлы .ouf.

Алгоритмы обхода дерева импорта широко используются в рекурсивных сборках и линковках при включении опции -r.

12. Динамически загружаемые модули

МультиОберон реализует динамическую загрузку модулей для всех вышеупомянутых бэкендов. Это может быть сделано следующими путями:

```
>Binwe\ombc run OmtestHelloWorld
>Binwe\ombsh OmtestHelloWorld
>Binwe\omlsh OmtestHelloWorld -ext bc
>Binue/ombc run OmtestHelloWorld
>Binue/ombsh OmtestHelloWorld
>Binue/omlsh OmtestHelloWorld -ext bc
```

Первый и второй варианты реализуются для всех платформ всех бэкендов, последний – только для Oml LLVM. Динамически загружаемый модуль должен быть в подготовленной бинарной форме согласно таблице ниже.

	Windows	Unix	Unix on Arm
Omb	.ocf BlackBox	.ocf BlackBox	--
Omf	.o COFF-format	.o ELF-format	.o ELF-format
Oml	.o COFF-format	.o ELF-format	.o ELF-format
Oml -ext bc	.bc LLVM	.bc LLVM	.bc LLVM

Файлы .ocf и .bc files создаются при компиляции.

Объектные файлы .o создаются при сборке внешним билдером.

Использование .bc файлов LLVM активирует компилятор LLVM JIT, который гораздо тяжелее, чем обычная загрузка объектника. Таким образом, использование JIT представляется непрактичным и слишком ресурсо-емким.

Для реализации всех упомянутых свойств, базовый модуль загрузчика Baseloader реализован в System. Модули, расширяющие тип загрузчика OmcOcfLoader, OmcObjLoader__Coff, OmcObjLoader__Elf, и OmlBcLoader реализуют специфическую функциональность загрузки ocf, coff, elf и bc.

13. Модуль Runner

МультиОберон использует специальный модуль Runner с вариациями для различных сред. Процедура Runner.SetRun регистрирует функцию MAIN которая должна быть вызвана после загрузки оболочки.

```
MODULE OmtestMkTraps;
  IMPORT Runner, OLog, SYSTEM;
  PROCEDURE MAIN*;
    VAR ar: Runner.ArgsReader; str: Runner.SName;
  BEGIN
    IF ~ar.StringOpt("-trap", str) THEN
      OLog.String("usage: "); OLog.SString(Runner.argv0);
      OLog.String(" -trap"); OLog.Ln;
      OLog.Tab; OLog.String("where -trap is as following:"); OLog.Ln;
      OLog.Tab; OLog.String("a - assert"); OLog.Ln;
      OLog.Tab; OLog.String("h - halt"); OLog.Ln;
      OLog.Tab; OLog.String("z - zero divide"); OLog.Ln;
      OLog.Tab; OLog.String("p - nil pointer dereference"); OLog.Ln;
    ELSE
      RunOpt(str);
    END;
  END MAIN;

BEGIN
  Runner.SetRun(MAIN)
END OmtestMkTraps.
```

Runner также содержит разбор опций командной строки процедурами Runner.ArgsReader.StringOpt(), Runner.ArgsReader.IntOpt().

Константы модуля Runner определяют величины, специфические для разных платформ:
Runner.RUN_TIME = "OMB" | "OMF" | "OML" для BlackBox, Ofront, LLVM сред.
Runner.OS_NAME = "Windows" | "Unix"
Runner.BIN_BITS = 64 | 32
Runner.KERNEL_VERSION = 16 | 17 | 18 для BlackBox Kernel
Runner.DESC_MACH = "X86" | "X64" | "ARM32" | "ARM64"

14. Возможности тестирования

Тестовые возможности покрывают как отдельные модули, так и компилятор в целом. Например, тестируется модуль OmtestSimple с процедурой PROCEDURE Sum(x, y: INTEGER): INTEGER. Нам потребуется модуль [module_name]_test – OmtestSimple_test.

```
MODULE OmtestSimple_test;
  IMPORT T := Testing, OmtestSimple;
  PROCEDURE Test0Basic* (VAR rec: T.Rec);
  BEGIN
    CASE rec.n_test OF
      | 0:
        rec.res_type := T.RES_INT;
        rec.msg := 'Sum x+x';
        rec.i_req := 6;
        rec.i_res := OmtestSimple.Sum(3, 0);
      | 1:
        rec.res_type := T.RES_INT;
        rec.msg := 'Sum x+y';
        rec.i_req := 5;
        rec.i_res := OmtestSimple.Sum(3, 2);
      | 2:
        rec.finish := TRUE;
    ELSE
      END;
  END Test0Basic;
END OmtestSimple_test.
```

Модуль OmtestSimple_test реализует набор процедур с шаблоном Test*, вызываемых тестовым окружением. Соглашение по именам Test0*, Test1* используется для вывода результатов тестирования в отсортированном по возрастанию порядке.

Каждый тест устанавливает поля структуры Testing.Rec. Тип возвращаемого значения res_type устанавливается первым, чтобы показать, какие поля используются. Сообщение пользователю msg выводится при тестировании. Для каждого типа результат сравнивается с требуемой величиной. В нашем случае i_res представляет собой целочисленный результат, i_req является целочисленной требуемой величиной. Тестовое окружение обнуляет структуры перед каждым вызовом, только номер сета и номер теста (n_test) устанавливаются предварительно. В случае несоответствия сообщение об ошибке выводится тестовой системой.

Тесты компилятора могут загружаться и выполняться динамически. Так что модули должны быть предварительно подготовлены для динамической загрузки:

```
>Binwe\ombc co -odc OmtestSimple OmtestSimple_test
>Binue\ombc co -odc OmtestSimple OmtestSimple_test
```

После компиляции используйте команду test в консольном компиляторе:

```
>Binwe\ombc test OmtestSimple
>Binue\ombc test OmtestSimple
[ALL] ===== Total 2 tests, 0 bad, result= 100.0%
```

Опция уровня печати -pl дает больше информации:

```
>Binwe\ombc test -pl 3 OmtestSimple
>Binue\ombc test -pl 3 OmtestSimple
```



```

[OmtestSimple_test.Test0Basic.000] INT i_res= 6 i_req= 6 :Sum x+x
[OmtestSimple_test.Test0Basic.001] INT i_res= 5 i_req= 5 :Sum x+y
[OmtestSimple_test.] ----- In module 1 sets, 2 tests, 0 bad
[ALL] ===== Total 2 tests, 0 bad, result= 100.0%

```

Специальные тесты для компилятора также прилагаются:

```

B\bwe_tests_tomake.bat      (* X86 *)
B\bwr_tests_tomake.bat      (* X64 *)
B/bue_tests_tomake.sh       (* X86 *)
B/bur_tests_tomake.sh       (* X64 *)
B/bu4_tests_tomake.sh       (* ArmV71 *)
B/bu8_tests_tomake.sh       (* Aarch64 *)

```

Компилируемые тесты ниже (OmtestOmc*) не относятся к специфическим модулям, а к компилятору в целом:

```

Binwe\ombc co -odc OmtestOmcSimple_test OmtestOmcStrings_test
OmtestOmcSystem_test OmtestOmcImports_test OmtestOmcExtensions_test
OmtestOmcBound_test OmtestOmcAdvanced_test

```

Специальные тесты компилятора также имеют скрипт сборки. Выполнение их приводит к печати, аналогичной следующей:

```

bwe_tests_run.bat
bue_tests_run.sh
c:\suok5\dsu>Binwe\ombc test -pl 2 OmtestOmcSimple_test
OmtestOmcStrings_test OmtestOmcSystem_test OmtestOmcImports_test
OmtestOmcExtensions_test OmtestOmcBound_test OmtestOmcAdvanced_test
[OmtestOmcSimple_test.] ----- In module 9 sets, 104 tests, 0 bad
[OmtestOmcStrings_test.] ----- In module 5 sets, 64 tests, 0 bad
[OmtestOmcSystem_test.Test1Addr.028] ?LONG li_res= 12 li_req= 16 :Proper
position of ptr after long
[OmtestOmcSystem_test.Test1Addr.029] ?LONG li_res= 16 li_req= 24 :SIZE of
Rec with LONGINT aligned by 8
[OmtestOmcSystem_test.Test1Addr.030] ?LONG li_res= 12 li_req= 16 :Proper
position of ptr after Rec with long
[OmtestOmcSystem_test.Test1Addr.031] ?LONG li_res= 12 li_req= 16 :SIZE of
Rec with LONGINT and char
[OmtestOmcSystem_test.Test1Addr.032] ?LONG li_res= 16 li_req= 24 :SIZE of
Rec with Rec with LONGINT aligned by 8
[OmtestOmcSystem_test.] ----- In module 3 sets, 49 tests, 5 bad
[OmtestOmcImports_test.] ----- In module 6 sets, 59 tests, 0 bad
[OmtestOmcExtensions_test.] ----- In module 10 sets, 80 tests, 0 bad
[OmtestOmcBound_test.] ----- In module 6 sets, 95 tests, 0 bad
[OmtestOmcAdvanced_test.] ----- In module 6 sets, 42 tests, 0 bad
[ALL] ===== Total 493 tests, 5 bad, result= 98.98580121703854%

```

Тесты для JIT-компилятора для Oml LLVM могут запускаться скриптами:

```

lwe_tests_jit_run.bat
lue_tests_jit_run.sh

```

Приводимые тесты компилятора также могут быть статически собраны в приложение OmtestOmcCompiler и запущены:

```

Omtest\Cfwe\OmtestOmcCompiler -pl 2
Omtest/Cfue/OmtestOmcCompiler -pl 2

```

Список непрошедших тестов демонстрирует качество специфического бэкэнда. Это используется для дальнейшего усовершенствования компилятора.

15. Возможности бенчмаркинга

Данные функции используются для измерений времени работы выбранных процедур. Требуется модуль [module_name]_bench – например, OmtestBenchRoutines_bench для оценки OmtestBenchRoutines.

```
MODULE OmtestBenchRoutines_bench;
  IMPORT T := Testing, OmtestBenchRoutines;
  PROCEDURE BenchmarkPalindrome* (IN bench: T.Bench; VAR num_done: INTEGER);
    VAR count: INTEGER;
  BEGIN
    num_done := 0; count := 0;
    WHILE num_done < bench.num DO
      IF OmtestBenchRoutines.IsPalindrome(
        "A man, a plan, a canal: Panama") THEN
        INC(count)
      END;
      INC(num_done)
    END
  END BenchmarkPalindrome;
END OmtestBenchRoutines_bench.
```

Модуль OmtestBenchRoutines_bench реализует набор процедур Benchmark*, вызываемых тестовым окружением. Каждая такая процедура на вход получает Testing.Bench. Параметр bench.num устанавливает требуемое число итераций. Выходной параметр num_done устанавливает реальное число итераций для контроля во избежании оптимизации реального числа итераций выполнения.

Временные тесты могут загружаться и выполняться динамически. Модули должны быть предварительно подготовлены для динамической загрузки:

```
>Binwe\ombc co -odc OmtestBenchRoutines OmtestBenchRoutines_bench
>Binue/ombc co -odc OmtestBenchRoutines OmtestBenchRoutines_bench
```

После компиляции используется команда bench в консоли компилятора:

```
>Binwe\ombc bench OmtestBenchRoutines
>Binue/ombc bench OmtestBenchRoutines
[OmtestBenchRoutines_bench.BenchmarkPalindrome] 1000000
00:00:00.282000 282.0 ns/op
```

Тестовое окружение информирует, что весь 1000000 итераций был выполнен за 282 миллисекунды. Среднее время составило 282.0 наносекунд на операцию:

16. Разработка платформо-зависимых модулей

МультиОберон предназначен для мульти-платформенной разработки. Процедуры одних модулей не зависят от платформо-специфических характеристик, а для других зависимость существует. Эти характеристики включают в себя интерфейсы библиотечных функций операционных систем, различные структуры данных для 32 и 64-битных реализаций, специфических интерфейсов модуля Kernel.

Каждый исходник модуля может состоять из специфических версий, помеченных ДВУМЯ ПОДЧЕРКИВАНИЯМИ как [module]__[specific]. Имена модулей в МультиОбероне не могут содержать 2 символа подчеркивания внутри или один на конце, иначе возникает ошибка "string expected". Однако, имена файлов могут иметь подчеркивания и специфические расширения для модулей. Более того, наличие подчеркивания в имени файла означает платформо-зависимую реализацию модуля. Например, есть две абсолютно разные реализации модуля OmcObjLoader:

- OmcObjLoader__Coff – для Windows формата COFF;
- OmcObjLoader__Elf – для Unix формата ELF.

Так что OmcObjLoader__Coff компилируется в OmcObjLoader.o под Windows в одном скрипте и OmcObjLoader__Elf компилируется в OmcObjLoader.o под Unix в другом скрипте.

Очевидно платформо-зависимые модули имеют отличающиеся интерфейсы. Компиляция платформо-независимого модуля:

```
>Binwe\ombc co -odc OmcDiscomp
omb:compiling c:\suok5\dsu\Omc\Mod\Discomp.odc
new symbol file >c:\suok5\dsu\Omc\Code\Discomp.ocf code=3652 glob=8716
Располагаются они для BlackBox в Omc/Sym и Omc/Code для кода.
```

Компиляция OmcObjLoader__Coff означает платформо-зависимость:

```
>Binwe\ombc co -odc OmcObjLoader__Coff
omb:compiling c:\suok5\dsu\Omc\Mod\ObjLoader__Coff.odc
new symbol file >c:\suok5\dsu\Omc\Cbwe\ObjLoader.ocf code=18480 glob=32
Платформо-зависимые для BlackBox будут в Omc/Sbwe и Omc/Cbwe для кода.
```

Но даже независимые модули Ofront и LLVM разделяются по каталогам:

```
>Binwe\omfc co -odc OmcDiscomp
omf:compiling c:\suok5\dsu\Omc\Mod\Discomp.odc
>c:\suok5\dsu\Omc\Cfwe\OmcDiscomp .c=26490 .h=3555
Платформо-независимые для Ofront будут в Omc/Sfwe и Omc/Cfwe для кода.
```

Большая часть платформо-зависимых модулей расположена в System и Host. Они компилируются на этапе инсталляции отдельно от пользовательского процесса разработки. Соглашение по именам использует следующие буквы в МультиОбероне:

- b-BlackBox, f-Ofront, l-LLVM;
- w-Windows, u-Unix;
- e-X86, r-X64;
- 16-BlackBox 1.6, 17-BlackBox1.7.

Модуль Runner требует 19 платформо-зависимых реализаций кода.

	Omb	Omf32	Omf64	Oml32	Oml64
BlackBox17Win	Runner__bwe17	Runner__fwe17	Runner__fwr17	Runner__lwe17	Runner__lwr17
BlackBox16Win	Runner__bwe16	Runner__fwe16	Runner__fwr16	Runner__lwe16	Runner__lwr16
BlackBox17Unix	Runner__bue17	Runner__fue17	Runner__fur17	Runner__lue17	Runner__lur17
17 ArmV71	--	Runner__fu4	--	Runner__lu4	--
17 Aarc64	--	--	Runner__fu8	--	Runner__lu8

Это очень неудобно разрабатывать все 19 платформо-зависимых модуля в виде исходников, поэтому специальные средства были разработаны, о которых в следующей главе.

17. Препроцессор модулей дженериков

Дженерики как понятие не включены в компилятор МультиОберона. Равно как и не планируется их включение в будущем. Однако, минимальная утилита конвертации .odc включена в систему Омс.

Модуль ОмсПрер представляет собой текстовый препроцессор для конвертации из дженериков в специфические модули.

По конвенции МультиОберона, дженерики расположены в каталоге GMod, специфические модули расположены в каталоге Mod. System/Docu/Quick-Start.odc демонстрирует, как генерятся специфические модули. Например, команда генерации Runner__fwr17 из GMod/Runner выглядит следующим образом:

```
^Q ОмсПрер.ToOdcFileList('OMF WIN V64 BB17', 'System/Mod')"  
@Омс/Mod/Defs.odc System/GMod/Runner.odc:Runner__fwr17
```

Сначала устанавливаются параметры окружения: OMF, WIN, V64, and BB17. Далее загружается файл определений Defs со специфическими установками, как-то @ADDR=LONGINT (это 64-бит, V64 установлено как параметр окружения). Далее GMod/Runner трансформируется в Runner__fwr17.

```
#IF @BB  
    SP = 4; DLT_STACK = 256;  
    RUN_TIME* = "OMB";  
#ELSIF @OMF  
    RUN_TIME* = "OMF";  
#ELSIF @OML  
    RUN_TIME* = "OML";  
#END  
    SysTrapProc = PROCEDURE (n: INTEGER; stpa: @ADDR);  
Результатом трансформации для модуля Runner__fwr17 будут:  
    RUN_TIME* = "OMF";  
    SysTrapProc = PROCEDURE (n: INTEGER; stpa: LONGINT);
```

Файл определений Defs содержит специальные настройки для всех необходимых платформ. Используются только макро-команды условной компиляции и механизм подстановки имен.

18. Ограничения

МультиОберон это масштабируемая технология на основе систем ограничений с начальной точкой в виде синтаксиса Компонентного Паскаля. Оператор RESTRICT используется для активации/деактивации основных языковых средств Оберона. Подсистема Restrict содержит набор специальных профилей. Модуль RestrictAdrint просто включает ADRINT как целое размерностью адреса:

```
RESTRICT +ADRINT*;
```

Символ “*” означает, что данное ограничение экспортируется в модуль, который непосредственно импортирует RestrictAdrint.

RestrictOberon07 profile uses more complex rules, like:

```
RESTRICT -CLOSE*, -EXIT*, -LOOP*, -OUT*, -WITH*,
        -PROCEDURE (PROCEDURE)*,      (* Recursion *)
        -BEGIN (PROCEDURE)*,          (* Nested Procedures *)
        -RETURN (PROCEDURE)*;         (* Single Return in the End only *)
```

Тест OmtestOmcRestrict_test демонстрирует использование ADRINT. Переменная типа ADRINT может сохранять адрес в качестве целочисленной величины.

```
IMPORT Api, RestrictAdrint;

PROCEDURE Xxx;

    VAR ai: ADRINT;

BEGIN

    ai := SYSTEM.VAL(ADRINT, Api.TestGetAdr());

END Xxx;
```

Тест OmtestOmcRestrict_test компилируется и выполняется:

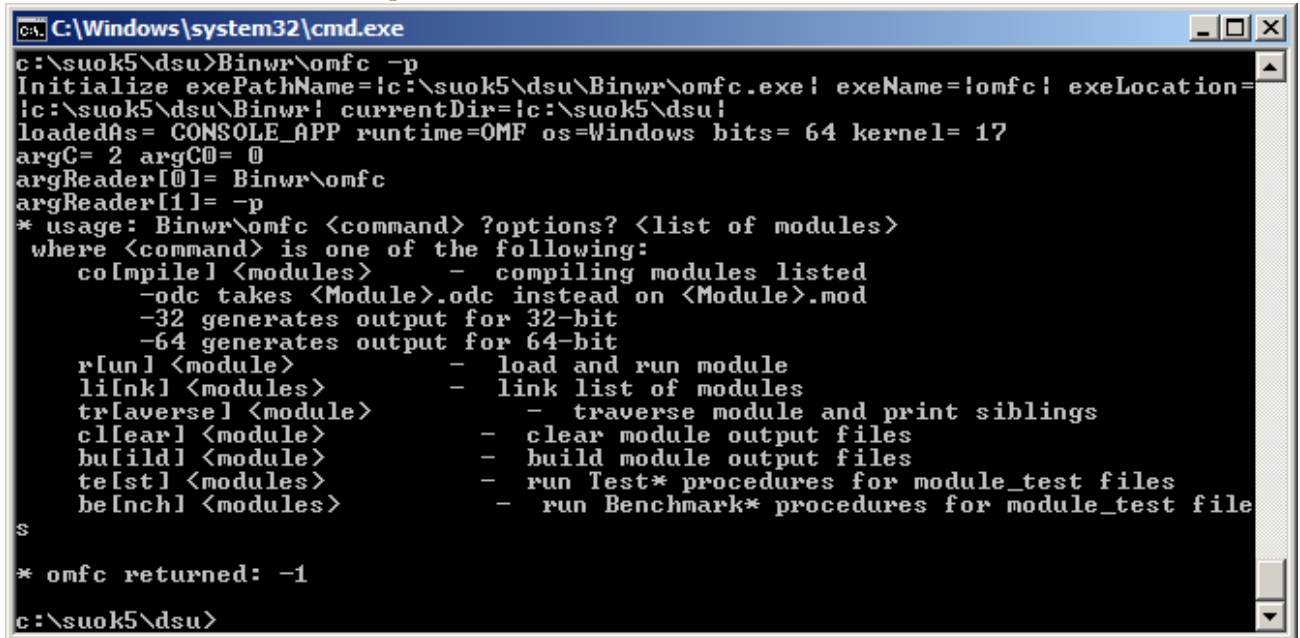
```
>Binwe\ombc co -odc OmtestOmcRestrict_test
>Binue/ombc co -odc OmtestOmcRestrict_test
>Binwe\ombc test -pl 2 OmtestOmcRestrict_test
>Binue/ombc test -pl 2 OmtestOmcRestrict_test
[ALL] ===== Total 14 tests, 0 bad, result= 100.0%
```

Дополнительная функциональность RESTRICT+ реализована пока только для ADRINT.

19. Команды и опции командной строки

Компилятор МультиОберона `om[backend]sh` без аргументов осуществляет печать подсказки использования. Опция `-p` печатает константы и глобальные переменные модуля Runner, помимо печати подсказки.

```
> Binwr\omfc co -p
```



```
C:\Windows\system32\cmd.exe
c:\suok5\dsu>Binwr\omfc -p
Initialize exePathName=!c:\suok5\dsu\Binwr\omfc.exe! exeName=!omfc! exeLocation=
!c:\suok5\dsu\Binwr! currentDir=!c:\suok5\dsu!
loadedAs= CONSOLE_APP runtime=OMF os=Windows bits= 64 kernel= 17
argC= 2 argC0= 0
argReader[0]= Binwr\omfc
argReader[1]= -p
* usage: Binwr\omfc <command> ?options? <list of modules>
  where <command> is one of the following:
    compile <modules>          - compiling modules listed
      -odc takes <Module>.odc instead on <Module>.mod
      -32 generates output for 32-bit
      -64 generates output for 64-bit
    run <module>                - load and run module
    link <modules>              - link list of modules
    traverse <module>           - traverse module and print siblings
    clear <module>              - clear module output files
    build <module>              - build module output files
    test <modules>              - run Test* procedures for module_test files
    bench <modules>            - run Benchmark* procedures for module_test file
s
* omfc returned: -1
c:\suok5\dsu>
```

Используются следующие опции командной строки:

- odc – брать .odc вместо .mob;
- 32 – 32-бит режим;
- 64 – 64-бит режим;
- os – Windows|Linux используется для кросс компиляции;
- r – для всего рекурсивного импорта;
- h – не включать HostConLog;
- n – перекомпилировать только новые файлы;
- tl – уровень трассировки (0-4);
- pl – уровень печати (0-3);
- ht – тип обработчика ошибок: 1-dlink, 2-frame pointer, 3-stack analysis;
- wsd – писать в каталог System.

Эти опции могут быть расширены опциями динамически загружаемых модулей. Последние обрабатываются в загружаемых модулях, а не в компиляторе.

20. Журнал изменений

may 2019 original MultiOberon pre-version 0.8 released

nov 2019 MultiOberon pre-version 0.9 released

jun 2020 MultiOberon pre-version 0.95 released

nov 2020 MultiOberon version 1.0 released – реализована вся базовая функциональность

feb 2021 MultiOberon version 1.1 released – поддержка ARM32 и ARM64

jul 2021 MultiOberon version 1.2 released

Use it and enjoy! - Ўъsalos y disfrъtalos! - Bonne utilisation - Приятного использования -
Powodzenia - Viel SpaЯ

Дагаев Дмитрий Викторович

dvdagaev@oberon.org