МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

САНКТ**-**ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**,** МЕХАНИКИ И ОПТИКИ

Кафедра Информатики и прикладной математики

Лабораторная работа №5 (8 вариант)

по предмету «Верификация моделей программ»

Выполнил:

студент гр. № Р4115

Назукин Д. Е.

Проверил :

Кореньков Ю. Д.

Санкт-Петербург

2017

**Цели**

Освоение верификации корректности гетерогенной программы методом интерпретации.

**Задачи**

Воспользовавшись созданной в предыдущем задании библиотекой, создать программу, принимающую в качестве аргумента командной строки имя файла, содержащего бинарное представление кода программы на исходном языке, загружающую этот файл и выполняющую сохранённую в нём программу. Информацию о точке входа получать аргументом командной строки или из информации в бинарном файле. Взаимодействие загруженного и выполняемого кода с остальной компьютерной системой реализовать через поддержку импортирования произвольных, задаваемых во входном коде, функций внешних DLL, используя функции ввода-вывода WinApi.

**Описание работы**

В данной работе будет реализован интерпретатор байт-кода из задания № 3. Интерпретатор (interpreter) — [программа](http://megabook.ru/article/%d0%9f%d1%80%d0%be%d0%b3%d1%80%d0%b0%d0%bc%d0%bc%d0%b0%20(%d0%ba%d0%be%d0%bc%d0%bf%d1%8c%d1%8e%d1%82%d0%b5%d1%80%d1%8b%20%d0%b8%20%d0%b8%d0%bd%d1%82%d0%b5%d1%80%d0%bd%d0%b5%d1%82)) или техническое средство, выполняющее интерпретацию, а также вид транслятора, осуществляющего пооперационную (покомандную) обработку и выполнение исходной программы или запроса. В отличие от компилятора, который осуществляет трансляцию всей программы высокого уровня в машинные коды один раз без ее выполнения (создает объектную программу), интерпретатор транслирует исходную программу команда за командой каждый раз при выполнении и не создает объектного модуля.

Также будет реализована поддержка импортирования произвольных, задаваемых во входном коде, функций внешних DLL. DLL — это библиотека содержащая код и данные, которые могут использоваться более чем одной программой одновременно.

**Аспекты реализации**

Интерпретатор работает следующим образом:

1. Считывает текущую команду
2. Выполняет текущую команду
3. Переходит к следующей команде. Затем шаг 1, пока не достигнется завершающая команда.

**while** (**true**) {  
 Command command = **commands**.get(**currentCommand**);  
 **if** (command.getName().equals(**"END"**)) {  
 **break**;  
 } **else** {  
 executeCommand(command);  
 }  
}

**private void** executeCommand(Command command) {  
 **switch** (CommandType.*valueOf*(command.getName())) {  
 **case *ADD***:  
 addCommand(command.getArgs());  
 **currentCommand**++;  
 **break**;  
 **case *MINUS***:  
 minusCommand(command.getArgs());  
 **currentCommand**++;  
 **break**;

. . .

Для поддержки импортирования внешний функций из DLL была написана *С* функция которая вызывается через *JNI*.

**public native** AbstractValue callNativeFunc(String lib, String funcName, AbstractValue[] args, String[] argsType, String returnType);

Для выполнения произвольных функции из DLL по их сигнатуре была использована библиотека *libffi.* Зная адрес функции (через *GetProcAddress*), типы аргументов (массив *ffi\_argTypes*) и тип возвращаемого значения (*c\_retType*), можно вызвать эту функцию передав указатели на аргументы в виде массива *void\* (values).* Результат выполнения будет в переменной *rc.*

ffi\_arg rc; *// return value*

**if** (ffi\_prep\_cif(&cif, FFI\_DEFAULT\_ABI, length, c\_retType, ffi\_argTypes) == FFI\_OK) {  
  
 ffi\_call(&cif, FFI\_FN(GetProcAddress(dllHandle, funcC)), &rc, values);  
} **else** {  
 **return** NULL;  
}

**Результаты**

Вход программы файл с текстом:

declare function CreateFile(lpFileName as string,  
 dwDesiredAccess as int,  
 dwShareMode as int,  
 lpSecurityAttributes as int,  
 dwCreationDisposition as int,  
 dwFlagsAndAttributes as int,  
 hTemplateFile as int) as int  
 lib "kernel32.dll" alias "CreateFileA"  
  
declare function WriteFile(hFile as int,  
 lpBuffer as string,  
 nNumberOfBytesToWrite as int,  
 lpNumberOfBytesWritten as int,  
 lpOverlapped as int) as int  
 lib "kernel32.dll"  
  
declare function CloseHandle(hFile as int) as int lib "kernel32.dll"  
  
declare function lstrlen(hFile as string) as int lib "kernel32.dll" alias "lstrlenA"

function print(text as string, handle)  
 WriteFile(handle, text, lstrlen(text), 0, 0);  
end function  
  
function main()  
 f = CreateFile("out.txt", 0x40000000, 0, 0, 2, 0, 0);  
  
 a = new User(3, "Ivan");

print("User= " + a.toString() + "\n", f);

CloseHandle(f);  
  
 main = 0;  
end function

class User  
 private id, name

public function New(\_id as long, \_name as string)  
 id = \_id;  
 name = \_name;  
end function

public function toString()  
 toString = "id: " + getId() + " name: " + name;  
end function

end class

В результате будет создан файл out.txt и в него записан результат:

User= id: 3 name: Ivan

**Вывод**

В ходе работы реализована программа (интерпретатор), принимающая в качестве аргумента командной строки имя файла, содержащего бинарное представление кода программы на исходном языке, загружающую этот файл и выполняющую сохранённую в нём программу.

Также реализовано импортирование произвольных внешних функций из DLL.