Практические задания к лабораторным работам и курсовому проекту по предмету Верификация моделей программ

Задание 1

Цель – освоение верификации синтаксических конструкций текстовых представлений данных.

Реализовать построение по исходному файлу с текстом синтаксического дерева с узлами, соответствующими правилам варианта, задающего язык для анализа. Вывести полученное дерево в файл.

Задание 2

Цель – освоение верификации алгоритмической последовательности модели программы.

Используя анализатор от первого задания реализовать программу, принимающую в качестве входных аргументов командной строки список имен N входных файлов и путь к выходной директории. Для заданных входных файлов с текстами на соответствующем варианту языке программа должна формировать в выходной директории выходные файлы, содержащие представление графа потока управления, описывающего последовательность выполнения операций, заданных инструкциями и выражениями соответствующей части исходного текста. В той же директории должен формироваться дополнительный файл, содержащий представление графа вызовов для всей совокупности входных файлов, описывающее отношения между функциями/методами входных файлов в плане обращению их друг к другу по именам.

Задание 3

Цель – освоение верификации формулируемости алгоритма в терминах моделей вычисления.

Дополнить программу из второго задания построением линейного кода определённого в соответствии с вариантом задания вида на основе имеющегося графа потока управления. Полученный линейный код для функций или методов вывести в мнемонической форме в файлы в той же директории.

Задание 4

Цель – освоение верификации простых манипуляций со структурами данных с учетом модели типизации.

Создать библиотеку, позволяющую сохранять в бинарный файл и загружать из него набор закодированных линейным кодом в соответствии с предыдущим заданием функций или методов. Изменить имеющуюся программу так, чтобы она, кроме входных файлов и имени выходной директории, принимала через аргументы командной строки имя для выходного файла с бинарным представлением, формируемого с помощью созданной библиотеки.

Дополнить входной язык возможностью определения пользовательских типов данных, обладающих набором полей и методов.

Задание 5

Цель – освоение верификации корректности гетерогенной программы методом интерпретации.

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.

Задание 6

Цель – освоение верификации программ с учетом полиморфизма диспетчеризации.

Добавить в разработанный за предыдущие задания программный комплекс поддержку перегрузки функций/методов и поддержку полиморфизма подтипов с переопределением реализаций логики (опционально виртуальные методы).

Задание 7

Цель – освоение верификации программ с учетом параметрического полиморфизма типов.

Реализовать в подготовленном программном комплексе поддержку параметрического полиморфизма через шаблонизацию либо обобщения (см. вариант).

Задание 8

Цель – освоение ручной верификации корректности алгоритма программы.

Разработать клиент-серверный модуль отладчика, позволяющий отлаживать выполняемую ранее созданным программным комплексом программу. За поддержку взаимодействия с отлаживаемой стороной отвечает процесс, выполняющий загруженную из бинарного файла программу. За отлаживающую сторону отвечает процесс нового элемента программного комплекса – пользовательский интерфейс отладчика в форме консольного либо графического приложения.

Функциональность к реализации:

- Точки останова по позиции в исходном коде
- Выполнение “по шагам” через предложения и строки исходного кода с их индикацией
- Отслеживание стека вызовов
- Просмотр значений переменных и элементов экземпляров составных типов по их именам
- Просмотр информации о типах и методах по их именам и экземплярам
- Перехват начала работы программы
- Ведение отладочной сессии с сохранением и подгрузкой информации о точках останова между запусками

Защита курсовой работы

При необходимости осуществляется итоговая очная защита курсовой работы, состоящая из реализации и сдачи заданной конкретной верификации для одной из моделей разработанного программного комплекса. Например: контроль исходного кода на соответствие некоторым дополнительным критериям корректности, контроль некоторых формальных утверждений для бинарного представления, контроль некоторых параметров программы во время выполнения.

Рекомендации и разъяснения

В случае возникновения вопросов по ходу выполнения заданий обращаться к преподавателю.

Входные языки и типизация

Для вариантов с динамической типизацией части входного языка, задающие информацию о типах мест размещения значений (таких как локальные переменные), должны игнорироваться, кроме определения функций, импортируемых из DLL (задание 5). Допустимым считается полное отсутствие спецификаций типа во входном тексте, кроме вышеозначенного случая, спецификации наследования и создания экземпляра сложного типа данных. Места размещения значений в соответствии с правилами динамической типизации могут принимать экземпляры любых типов.

Для вариантов со статической типизацией спецификации типа во всех случаях обязательны. Места размещения значений могут принимать только экземпляры специфицированных в их определении типов, либо совместимых по присваиванию в соответствии с семантикой наследования.

Оформление отчетов

К концу семестра по каждой из работ должен быть представлен отчет, содержащие следующие части:

1. Цели
2. Задачи
3. Описание работы
4. Аспекты реализации
5. Результаты
6. Выводы

Варианты заданий

Номер в группе	Входной язык	Типизация	Байт-код	Параметризация
1	1	Статическая	Стековый	Обобщения
2	2	Статическая	Регистровый	Обобщения
3	3	Статическая	Стековый	Обобщения
4	4	Статическая	Регистровый	Обобщения
5	5	Статическая	Стековый	Обобщения
6	1	Динамическая	Регистровый	Обобщения
7	2	Динамическая	Стековый	Обобщения
8	3	Динамическая	Регистровый	Обобщения
9	4	Динамическая	Стековый	Обобщения
10	5	Динамическая	Регистровый	Обобщения
11	1	Статическая	Регистровый	Шаблонизация
12	2	Статическая	Стековый	Шаблонизация
13	3	Статическая	Регистровый	Шаблонизация
14	4	Статическая	Стековый	Шаблонизация
15	5	Статическая	Регистровый	Шаблонизация
16	1	Динамическая	Стековый	Шаблонизация
17	2	Динамическая	Регистровый	Шаблонизация
18	3	Динамическая	Стековый	Шаблонизация
19	4	Динамическая	Регистровый	Шаблонизация
20	5	Динамическая	Стековый	Шаблонизация

Входные языки

```
identifier: "[a-zA-Z_][a-zA-Z_0-9]*"; // идентификатор

str: "\"[^\"\\]*(?:\\.[^\"\\]*)*\""; // строка, окруженная двойными кавычками
char: "'[^']'";                      // одиночный символ в одинарных кавычках
hex: "0[xX][0-9A-Fa-f]+";            // шестнадцатеричный литерал
bits: "0[bB][01]+";                  // битовый литерал
dec: "[0-9]+";                       // десятичный литерал
bool: 'true'|'false';                // булевский литерал

list<item>: (item (',' item)*)?; // список элементов, разделённых запятыми
```

Вариант 1

```
source: sourceItem*;

typeRef: {
      |builtin: 'bool'|'byte'|'int'|'uint'|'long'|'ulong'|'char'|'string';
      |custom: identifier;
      |array: typeRef '[' (',')* ']';
};

funcSignature: typeRef? identifier '(' list<argDef> ')' {
      argDef: typeRef? identifier;
};

sourceItem: {
      |funcDef: funcSignature (statement.block|';');
};

statement: {
      |var: typeRef list<identifier ('=' expr)?> ';'; // for static typing
      |if: 'if' '(' expr ')' statement ('else' statement)?;
      |block: '{' statement* '}';
      |while: 'while' '(' expr ')' statement;
      |do: 'do' block 'while' '(' expr ')' ';';
      |break: 'break' ';';
      |expression: expr ';'
};

expr: { // присваивание через '='
      |binary: expr binOp expr; // где binOp - символ бинарного оператора
      |unary: unOp expr; // где unOp - символ унарного оператора
      |braces: '(' expr ')';
      |call: expr '(' list<expr> ')';
      |indexer: expr '[' list<expr> ']';
      |place: identifier;
      |literal: bool|str|char|hex|bits|dec;
};
```

Дополнения к 4 и 5 заданиям

```
sourceItem: {
      |funcDef: importSpec? funcSignature (statement.block|';') {
            importSpec: 'extern' '(' dllName (',' dllEntryName)? ')';
            dllName: str; // имя DLL, содержащей импортируемую функцию
            dllEntryName: str; // имя точки входа при отличии от объявляемого
      };
      |classDef: 'class' identifier '{' member* '}' {
            member: modifier? (funcDef|field);
            field: typeRef? list<identifier> ';';
            modifier: 'public'|'private';
      };
};
```

Вариант 2

```
source: sourceItem*;

typeRef: {
      |builtin: 'bool'|'byte'|'int'|'uint'|'long'|'ulong'|'char'|'string';
      |custom: identifier;
      |array: 'array' '[' (',')* ']' 'of' typeRef;
};

funcSignature: identifier '(' list<argDef> ')' (':' typeRef)? {
      argDef: identifier (':' typeRef)?;
};

sourceItem: {
      |funcDef: 'method' funcSignature (body|';') {
            body: ('var' (list<identifier> (':' typeRef)? ';')*)?
statement.block;
      };

statement: {
      |if: 'if' expr 'then' statement ('else' statement)?;
      |block: 'begin' statement* 'end' ';';
      |while: 'while' expr 'do' statement;
      |do: 'repeat' statement ('while'|'until') expr ';';
      |break: 'break' ';';
      |expression: expr ';';
};

expr: { // присваивание через ':='
      |binary: expr binOp expr; // где binOp - символ бинарного оператора
      |unary: unOp expr; // где unOp - символ унарного оператора
      |braces: '(' expr ')';
      |call: expr '(' list<expr> ')';
      |indexer: expr '[' list<expr> ']';
      |place: identifier;
      |literal: bool|str|char|hex|bits|dec;
};
```

Дополнения к 4 и 5 заданиям

```
sourceItem: {
      |funcDef: 'method' funcSignature (body|';'|importSpec) {
            body: varsSpec? statement.block;
            varsSpec: 'var' (list<identifier> (':' typeRef)? ';')*;
            importSpec: 'from' (dllEntryName 'in')? dllName ';';
            dllName: str; // имя DLL, содержащей импортируемую функцию
            dllEntryName: str; // имя точки входа при отличии от объявляемого
      };
      |classDef: 'class' identifier funcDef.varsSpec 'begin' member* 'end' {
            member: modifier? (funcDef);
            modifier: 'public'|'private';
      };
};
```

Вариант 3

```
source: sourceItem*;

typeRef: {
      |builtin: 'bool'|'byte'|'int'|'uint'|'long'|'ulong'|'char'|'string';
      |custom: identifier;
      |array: typeRef '(' (',')* ')';
};

funcSignature: identifier '(' list<argDef> ')' ('as' typeRef)? {
      argDef: identifier ('as' typeRef)?;
};

sourceItem: {
      |funcDef: 'function' funcSignature statement* 'end' 'function';
};

statement: {
      |var: 'dim' list<identifier> 'as' typeRef;// for static typing
      |if: 'if' expr 'then' statement* ('else' statement*)? 'end' 'if';
      |while: 'while' expr statement* 'wend';
      |do: 'do' statement* 'loop' ('while'|'until') expr;
      |break: 'break';
      |expression: expr ';';
};

expr: { // присваивание через '='
      |binary: expr binOp expr; // где binOp - символ бинарного оператора
      |unary: unOp expr; // где unOp - символ унарного оператора
      |braces: '(' expr ')';
      |callOrIndexer: expr '(' list<expr> ')';
      |place: identifier;
      |literal: bool|str|char|hex|bits|dec;
};
```

Дополнения к 4 и 5 заданиям

```
sourceItem: {
      |funcDef: 'function' funcSignature statement* 'end' 'function';
      |externFuncDef: 'declare' 'function' funcSignature 'lib' dllName
('alias' dllEntryName)? {
            dllName: str; // имя DLL, содержащей импортируемую функцию
            dllEntryName: str; // имя точки входа при отличии от объявляемого
      };
      |classDef: 'class' identifier member* 'end' 'class' {
            member: modifier? (funcDef|field|externFuncDef);
            field: list<identifier> ('as' typeRef)?;
            modifier: 'public'|'private';
      };
};
```

Вариант 4

```
sourceItem: {
      |funcDef: identifier '[' list<identifier> ']' ':=' statement;
};

statement: {
      |expression: expr ';';
};

expr: {
      |set: identifier '=' expr; // связывание значения выражения с именем
      |setDelayed: identifier argList? ':=' expr { // отложенное связывание
            argList: '[' list<identifier> ']';
      };
      |binary: expr binOp expr; // где binOp - символ бинарного оператора
      |unary: unOp expr; // где unOp - символ унарного оператора
      |braces: '(' expr ')';
      |call: expr '[' list<expr> ']';
      |place: identifier;
      |vector: '{' list<expr> '}'; // формирование экземпляра списка/кортэжа
      |apply: expr '@@' expr; // применение списка справа к функции слева
      |kvpair: expr '->' expr; // формирование key-value-пары
      |map: '<|' list<expr> '|>'; // формирование ассоциативного vector'а
      |lookup: expr '[[' list<span> ']]'; { // взятие среза vector'а
            span: expr (';;' expr (';;' expr)?)?; // from index, to, step
      };
      |literal: bool|str|char|hex|bits|dec;
};

// функция как объект первого порядка, связанный с символическим именем
// присваивание выполняет связывание значения с символическим именем,
// отложенное присваивание связывает выражение, возможно с аргументами,
// откладывая вычисление выражения до момента применения символа
//
// ветвление, блоки, области видимости, императивные
// примитивы и др. реализуются
// посредством функций вида If[condExpr, thenExpr, elseExpr], Block[{x}]
// ';' как оператор, по очереди выполняющий выражения по обе стороны
//     себя и возвращающий результат правого
//
//
```

Вариант 5

```
source: sourceItem*;

typeRef: {
      |builtin: 'bool'|'byte'|'int'|'uint'|'long'|'ulong'|'char'|'string';
      |custom: identifier;
      |array: typeRef 'array' '[' dec ']'; // число - размерность
};

funcSignature: identifier '(' list<arg> ')' ('of' typeRef)? {
      arg: identifier ('of' typeRef)?;
};

sourceItem: {
      |funcDef: 'def' funcSignature statement* 'end';
};

statement: { // присваивание через '='
      |if: 'if' expr 'then' statement ('else' statement)?;
      |loop: ('while'|'until') expr statement* 'end';
      |repeat: statement ('while'|'until') expr ';';
      |break: 'break' ';';
      |expression: expr ';';
      |block: ('begin'|'{') (statement|sourceItem)* ('end'|'}');
};

expr: {
      |binary: expr binOp expr; // где binOp - символ бинарного оператора
      |unary: unOp expr; // где unOp - символ унарного оператора
      |braces: '(' expr ')';
      |call: expr '(' list<expr> ')';
      |slice: expr '[' list<range> ']' { // индексация или срез массива
            ranges: expr ('..' expr)?; // from index, to
      };
      |place: identifier;
      |literal: bool|str|char|hex|bits|dec;
};
```

Дополнения к 4 и 5 заданиям

```
sourceItem: {
      |funcDef: 'def' funcSignature statement* 'end';
      |externFuncDef: 'import' funcSignature 'native' dllName ('entry'
dllEntryName)? {
            dllName: str; // имя DLL, содержащей импортируемую функцию
            dllEntryName: str; // имя точки входа при отличии от объявляемого
      };
      |classDef: 'class' identifier member* 'end' {
            member: modifier? (funcDef|field|externFuncDef);
            field: 'var' list<identifier> ('of' typeRef)? ';';
            modifier: 'public'|'private';
      };

};
```

Получение оценки

По мере освоения практических навыков по данной дисциплине, обучающийся выполняет практические задания и итеративно защищает их индивидуальным образом.

Разработанный в практических заданиях комплекс программ в совокупности представляет собой курсовой проект, который при необходимости так же итеративно дорабатывается до готовности, после чего защищается.

Следующая таблица показывает, на какую базовую оценку автоматом обучающийся может претендовать в зависимости от количества выполненных им к определённому сроку практических заданий. Дополнительно к базовой оценке обучающийся может сдать экзамен по теоретической части курса, тем самым повысив итоговый балл.

Вид	***Заданий в срок***	**До сессии**	**В сессию**	**После сессии**
Дисциплина	5	3		
	7	4	3	
	8	5	4	3
Курсовая	Без защиты	5	4	3
	Защита		5	4