Белорусский государственный университет информатики и радиоэлектроники

Кафедра информатики

Лабораторная работа № 4

**Интерпретатор**

Выполнил студент. гр. 453503: Езерская Е.Е.

Проверил ассистент КИ: Шиманский В.В.

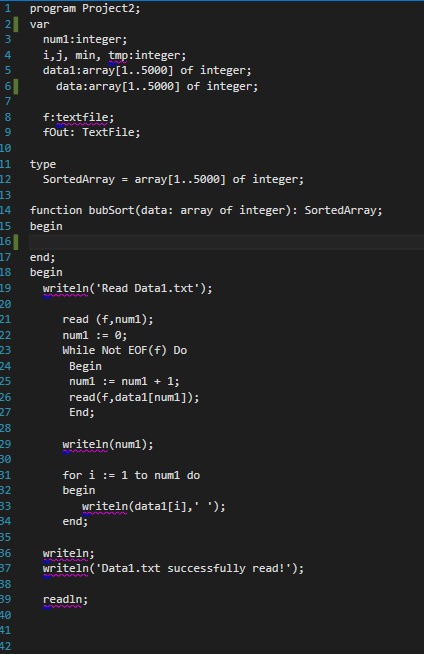
Минск, 2017

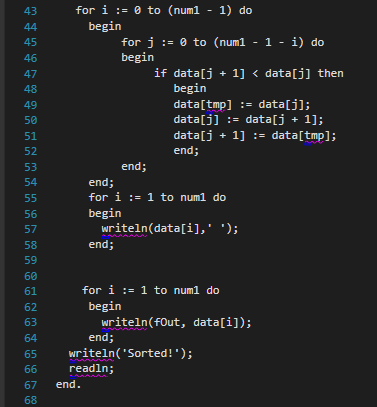
В данной работе ставится задача исследовать область интерпретаторов, рассмотреть существующие аналоги и написать свой собственный интерпретатор, на основе написанных трёх первых лабораторных, способный распознавать ошибки подмножества языка программирования, а также реализовать собственный компилятор, для создания байт-кода исходной программы.

В качестве анализируемого подмножества языка программирования будет использован язык программирования Pascal.

Для написания анализатора использован язык программирования C#.

На Рис. 1.1 показан код программы на языке Pascal.





**Рис. 1.1. Код программы на языке Pascal**

**2.Теория**

**Интерпрета́тор** — [программа](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) (разновидность [транслятора](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80)), выполняющая *интерпретацию*.

**Интерпрета́ция** — [пооператорный](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) (покомандный, построчный) анализ, обработка и тут же выполнение исходной программы или запроса (в отличие от [компиляции](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80), при которой программа транслируется без её выполнения).

Интерпретаторы — трансляторы языков программирования, работают на отличающемся от компиляторов принципе. Интерпретаторы не производят исполняемого машинного кода. Они берут исходный текст программы на языке программирования и выполняют его сами строка за строкой. При этом интерпретатор извлекает из файла с исходным текстом одну команду, распознает ее и вызывает те или иные функции операционной системы. Интерпретатор определяет команду и переводит (интерпретирует) ее так, чтобы операционная система поняла, что от нее хотят. Скорость выполнения программ в режиме интерпретации намного ниже, чем у компилированного кода, за счет того, что работа программы идет не напрямую с центральным процессором на языке машинных команд, а через программу-посредника, которая и тратит большое количество времени на распознавание исходного текста. В отличие от интерпретаторов, компиляторы «знакомятся» с исходными текстами программы всего один раз, когда делают из текста на языке программирования машинный код.

**Типы интерпретаторов**

**Простой интерпретатор** анализирует и тут же выполняет (собственно интерпретация) программу покомандно (или построчно), по мере поступления её исходного кода на вход интерпретатора. Достоинством такого подхода является мгновенная реакция. Недостаток — такой интерпретатор обнаруживает ошибки в тексте программы только при попытке выполнения команды (или строки) с ошибкой.

Простые интерпретаторы анализируют и выполняют (интерпретируют) программу последовательно (покомандно или построчно). Синтаксические ошибки обнаруживаются, когда интерпретатор приступает к выполнению команды (строки) содержащей ошибку. Сложные интерпретаторы компилирующего типа перед выполнением производят компиляцию исходного кода программы в машинный или «промежуточный код». Они быстрее выполняют большие и циклические программы, не занимаются анализом исходного кода в реальном времени. Некоторые интерпретаторы для начинающих программистов (преимущественно, для языка Бейсик) могут работать в режиме диалога, добавляя вводимую строку команд в программу (в памяти) или выполняя команды непосредственно.

**Интерпретатор компилирующего типа** — это система из [компилятора](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80), переводящего исходный код программы в промежуточное представление, например, в [байт-код](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82-%D0%BA%D0%BE%D0%B4) или [p-код](https://ru.wikipedia.org/wiki/P-%D0%BA%D0%BE%D0%B4), и собственно интерпретатора, который выполняет полученный промежуточный код (так называемая [виртуальная машина](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0)). Достоинством таких систем является большее быстродействие выполнения программ (за счёт выноса анализа исходного кода в отдельный, разовый проход, и минимизации этого анализа в интерпретаторе). Недостатки — большее требование к ресурсам и требование на корректность исходного кода. Применяется в таких языках, как [Java](https://ru.wikipedia.org/wiki/Java), [PHP](https://ru.wikipedia.org/wiki/PHP), [Tcl](https://ru.wikipedia.org/wiki/Tcl), [Perl](https://ru.wikipedia.org/wiki/Perl), [REXX](https://ru.wikipedia.org/wiki/REXX) (сохраняется результат [парсинга](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7) исходного кода[[5]](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B0%D1%82%D0%BE%D1%80#cite_note-5)), а также в различных [СУБД](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

В случае разделения интерпретатора компилирующего типа на компоненты получаются компилятор языка и простой интерпретатор с минимизированным анализом исходного кода. Причём исходный код для такого интерпретатора не обязательно должен иметь [текстовый формат](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82) или быть байт-кодом, который понимает только данный интерпретатор, это может быть [машинный код](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) какой-то существующей аппаратной платформы. К примеру, виртуальные машины вроде [QEMU](https://ru.wikipedia.org/wiki/QEMU), [Bochs](https://ru.wikipedia.org/wiki/Bochs), [VMware](https://ru.wikipedia.org/wiki/VMware) включают в себя интерпретаторы машинного кода [процессоров](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) семейства [x86](https://ru.wikipedia.org/wiki/X86).

Некоторые интерпретаторы (например, для языков [Лисп](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D1%81%D0%BF), [Scheme](https://ru.wikipedia.org/wiki/Scheme), [Python](https://ru.wikipedia.org/wiki/Python), [Бейсик](https://ru.wikipedia.org/wiki/%D0%91%D0%B5%D0%B9%D1%81%D0%B8%D0%BA) и других) могут работать в режиме диалога или так называемого цикла чтения-вычисления-печати. В таком режиме интерпретатор считывает законченную конструкцию языка (например, [s-expression](https://ru.wikipedia.org/w/index.php?title=S-expression&action=edit&redlink=1) в языке Лисп), выполняет её, печатает результаты, после чего переходит к ожиданию ввода пользователем следующей конструкции.

Уникальным является язык [Forth](https://ru.wikipedia.org/wiki/Forth), который способен работать как в режиме интерпретации, так и компиляции входных данных, позволяя переключаться между этими режимами в произвольный момент, как во время трансляции исходного кода, так и во время работы программ.

Следует также отметить, что режимы интерпретации можно найти не только в программном, но и [аппаратном обеспечении](https://ru.wikipedia.org/wiki/%D0%90%D0%BF%D0%BF%D0%B0%D1%80%D0%B0%D1%82%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5). Так, многие [микропроцессоры](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) интерпретируют машинный код с помощью встроенных [микропрограмм](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0), а процессоры семейства x86, начиная с [Pentium](https://ru.wikipedia.org/wiki/Pentium) (например, на архитектуре [Intel P6](https://ru.wikipedia.org/wiki/Intel_P6)), во время исполнения машинного кода предварительно транслируют его во внутренний формат (в последовательность микроопераций).

**Алгоритм работы простого интерпретатора**

1. прочитать инструкцию;
2. проанализировать инструкцию и определить соответствующие действия;
3. выполнить соответствующие действия;
4. если не достигнуто условие завершения программы, прочитать следующую инструкцию и перейти к пункту 2.

**Достоинства интерпретаторов**

* Бо́льшая [переносимость](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) интерпретируемых программ — программа будет работать на любой [платформе](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B0), на которой есть соответствующий интерпретатор.
* Как правило, более совершенные и наглядные средства диагностики ошибок в исходных кодах.
* Меньшие размеры кода по сравнению с машинным кодом, полученным после обычных компиляторов.

**Недостатки интерпретаторов**

* Интерпретируемая программа не может выполняться отдельно без программы-интерпретатора. Сам интерпретатор при этом может быть очень компактным.
* Интерпретируемая программа выполняется медленнее, поскольку промежуточный анализ исходного кода и планирование его выполнения требуют дополнительного времени в сравнении с непосредственным исполнением машинного кода, в который мог бы быть скомпилирован исходный код.
* Практически отсутствует [оптимизация кода](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D1%82%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), что приводит к дополнительным потерям в скорости работы интерпретируемых программ.

**3. Программа и комментарии**

**3.1. Область охватывания языка программирования написанным интерпретатором**

Данный интерпретатор включает в себя:

Типы данных:

* Var (статическое определение типа данных, таких как number, string, boolean, array)
* Function

Виды значений:

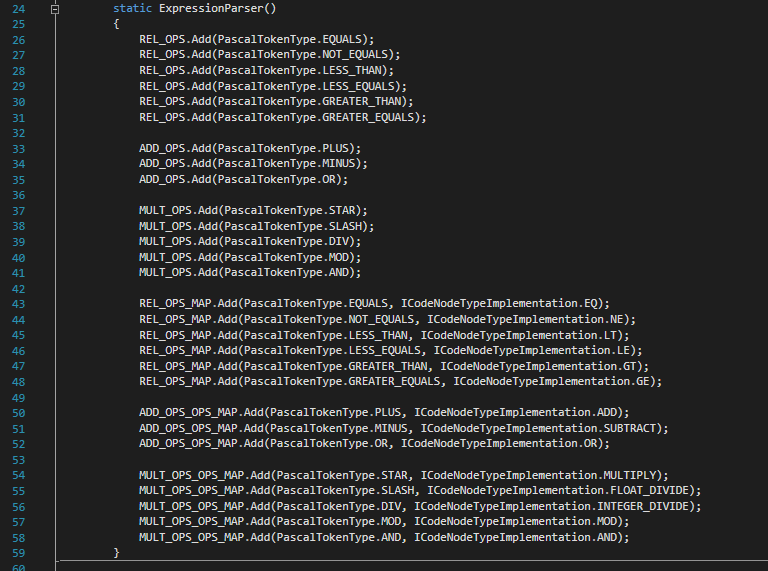
* integer
* bool
* Null

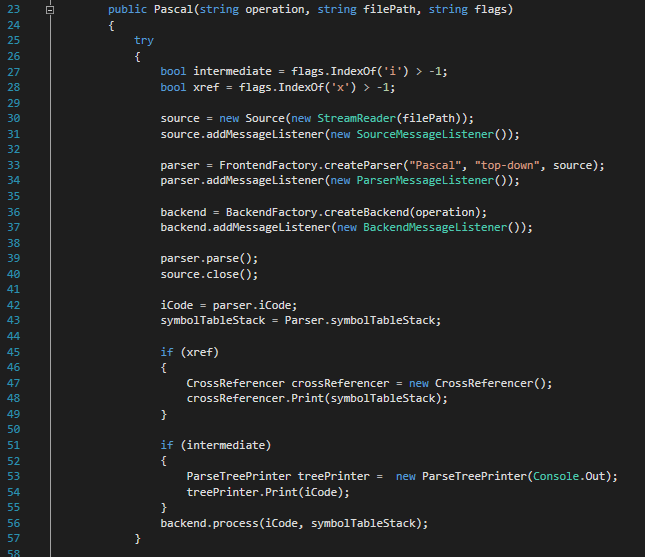
Виды операций:

* =
* +=
* -=
* \*=
* /=
* ||
* &&
* ===
* !==
* <
* <=
* >
* >=

Виды циклов и условные операторы:

* While
* If

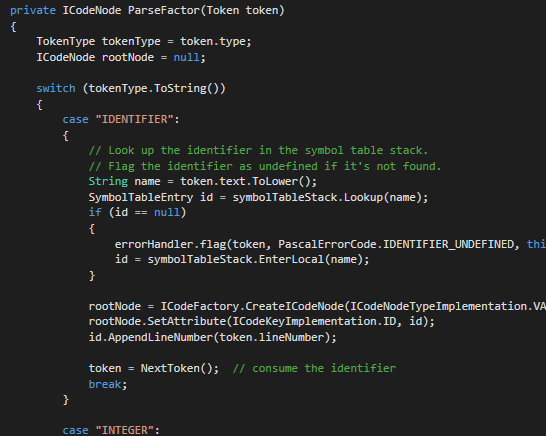
  
**Рис. 3.2.2. Фрагмент кода таблицы исполняемых команд**



**3.2. Главный модуль интерпретатора**

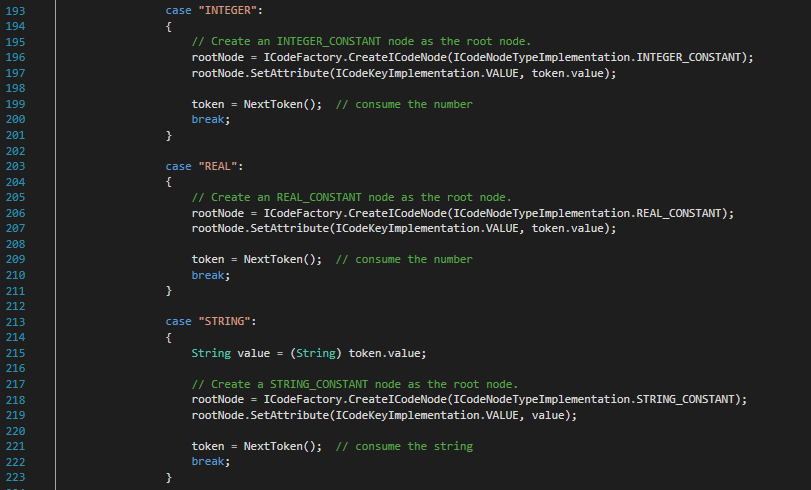
В результате выполнения работы использовались уже написанный лексический, синтаксический и семантический анализаторы. А также был реализован компилятор, для создания байт-кода. Который, с помощью интерпретатора, выполняется и приводит в жизнь код исходной программы.

Для построения таблицы байт-кода были использованы следующие фрагменты кода Рис. 3.2.1. - Рис. 3.2.3.

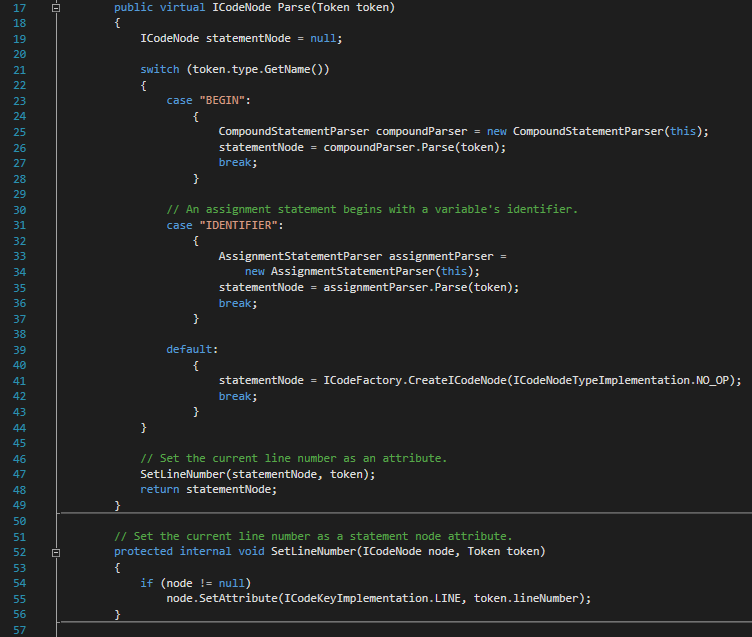


**Рис. 3.2.1. Фрагмент кода таблицы байт-кода**

Для построения компилятора были использованы следующие фрагменты кода Рис. 3.2.4. и Рис. 3.2.5.

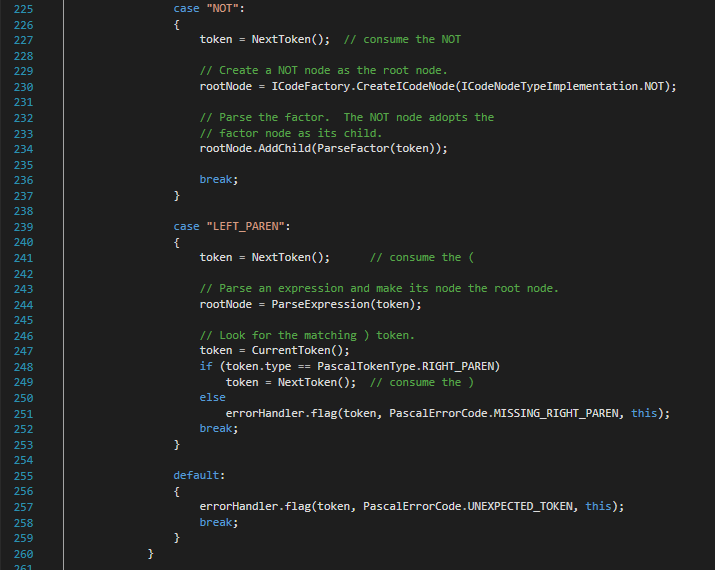


**Рис. 3.2.4. Фрагмент кода компилятора**

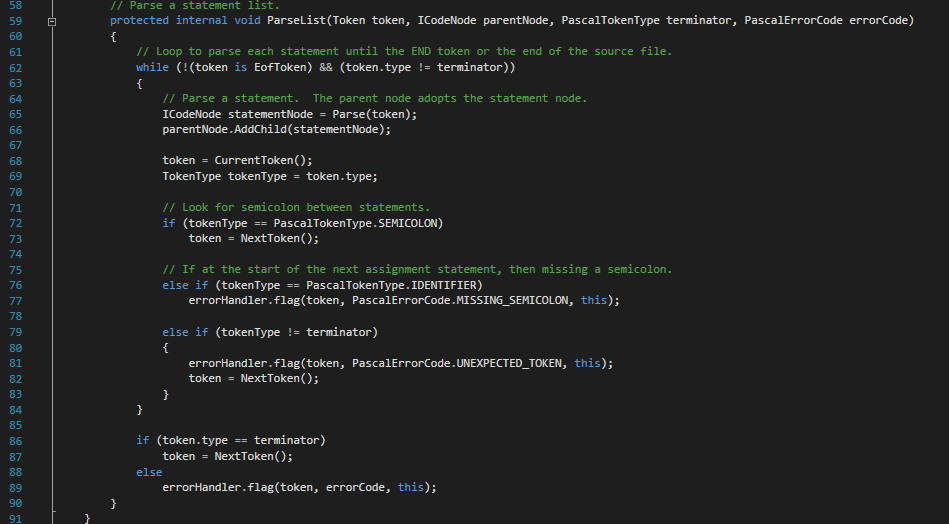


**Рис. 3.2.5. Фрагмент кода компилятора**

Для построения интерпретатора были использованы следующие фрагменты кода Рис. 3.2.6. и Рис. 3.2.7.



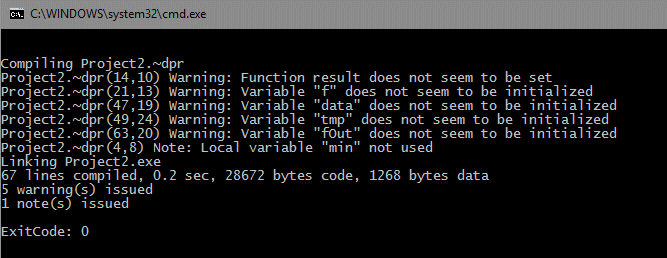
**Рис. 3.2.6. Фрагмент кода интерпретатора**



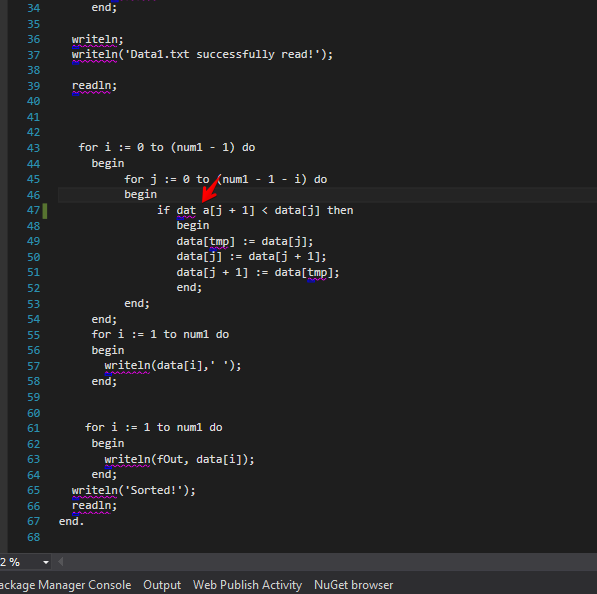
**Рис. 3.2.7. Фрагмент кода интерпретатора**

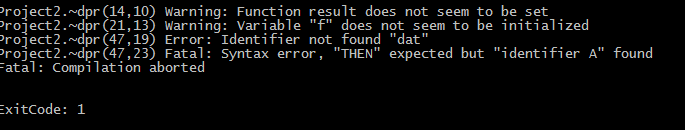
**4. Результаты выполнения**

Результат работы интерпретатора, по выполнению исходного кода программы Pascal без допущения ошибок, приведен на Рис. 4.1.

  
**Рис. 4.1. Результат работы интерпретатора без ошибок**

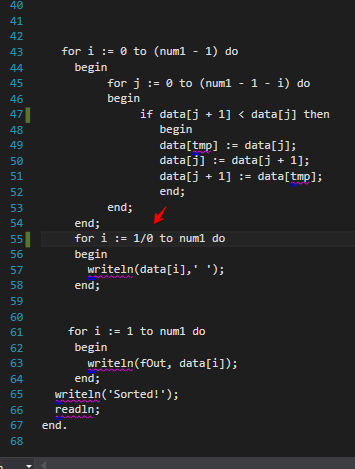
Также, при допущении ошибок в исходном коде, они выводятся на экран (выводится ошибка и описание ошибки).

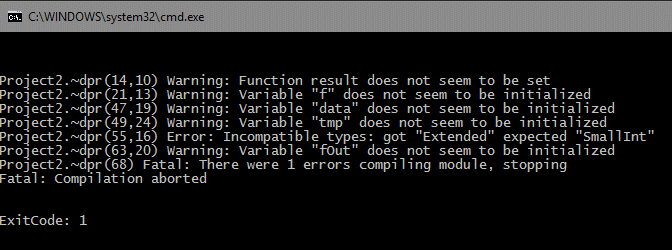
Результат работы интерпретатора, при допущении 1-ой ошибки (блок if не имеет закрывающую фигурную скобку) Рис. 4.6.  




**Рис. 4.6. Результат работы с 1-ой ошибкой**

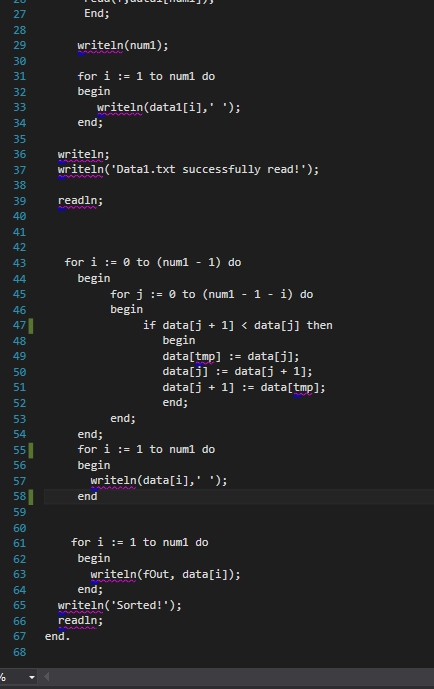
Результат работы интерпретатора, при допущении 2-ой ошибки (некорректная операция с переменной) Рис. 4.7.

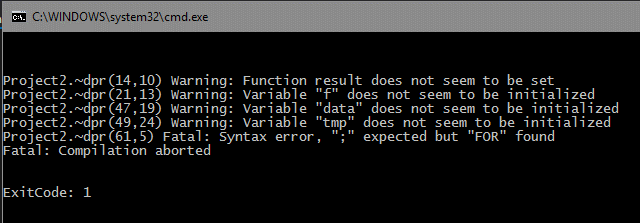




**Рис. 4.7. Результат работы со 2-ой ошибкой**

Результат работы интерпретатора, при допущении 3-ой ошибки (переменная не объявлена) Рис. 4.8.





**Рис. 4.8. Результат работы с 3-ей ошибкой**

**5. Выводы**

В результате выполнения работы были получены знания о интерпретаторах, компиляторах и их предназначении.

В итоге работы был построен свой собственный простой интерпретатор на основе уже имеющегося лексического, синтаксического и семантического для анализируемого подмножества языка программирования Pascal. Сам интерпретатор написан на языке программирования C#.

Данный интерпретатор способен обнаружить ошибки в коде, уведомлять о них, генерируя исключения. Также был реализован собственный компилятор, для получения байт-кода программы, который выполняется с помощью интерпретатора. Данный интерпретатор далек от совершенства, но способен выполнять простые программы на языке Pascal.

Было замечено, что при работе интерпретатора нужно непрерывно следить за ходом выполнения программы.