Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Расчетно-пояснительная записка

По курсу конструирование компиляторов

НА ТЕМУ:

Разработка интерпретатора языка LISP

Студент _____ИУ7-23М__ _____ **А.С. Камакин**
(Группа) (Подпись, дата) (И.О.Фамилия)

Преподаватель _____ **А.А. Ступников**
(Подпись, дата) (И.О.Фамилия)

2020 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

З А Д А Н И Е
по курсу конструирование компиляторов

Студент группы __ИУ7-23М___Камакин Андрей Сергеевич_____
(фамилия, имя, отчество)

Цель: овладение алгоритмами синтаксического анализа текста и навыками построения грамматик для разбора и аналитики входного текста.

Задачи:

- закрепление на практике имеющихся и приобретение новых специализированных знаний, умений, навыков и компетенций по конструированию компиляторов;
- анализ существующих проблем в выбранной предметной области и определение возможных путей для их решения;
- анализ существующего программного обеспечения(ПО) предметной области. Обзор методов, алгоритмов и программного обеспечения, которые могут быть использованы для решения выявленных проблем предметной области и разработки нового ПО;

Аналитический раздел

Основания для разработки

Разработка ведется в рамках проведения учебной практики по конструированию компиляторов на кафедре «Программное обеспечение ЭВМ и информационные технологии» факультета «Информатика и системы управления» МГТУ им. Н. Э. Баумана.

Назначение разработки

Построение минимального интерпретатора языка Lisp, реализующий только самые необходимые для работы функции. Предоставить возможность описывать новые функции и хранить их в памяти для их последующего использования.

Существующие аналоги

Сам язык Lisp появился в 1955 году и на текущий момент представлен огромным количеством диалектов и реализаций: Maclisp, Interlisp, Lisp Machine Lisp, Scheme, NIL, Franz Lisp, Common Lisp, Le Lisp, T, Emacs Lisp, AutoLISP, PicoLisp, EuLisp, ISLISP, OpenLisp, PLT Scheme, Racket, GNU Guile, Visual LISP, Qi (Qill), Shen, Clojure, Arc, LFE, Hy. Все вышеперечисленные реализации как-то изменяют грамматику языка и реализовывают ее на разных технологиях.

Описание системы

Проект должен представлять собой интерпретатор, который принимает на вход выражение, проводит над ним лексический анализ, строит синтаксическое дерево, вычисляет выражение и выводит результат пользователю. Пользователь имеет возможность не только вычислять простые выражения, но и определять свои функции для их последующего использования.

Общие требования к системе

1. Обработка ошибок, исключения и ошибки не приводят к зависанию или падению процесса;

2. Возможность вычисления выражений пользователя;

Введение

С начала 50-х годов XX века создаются и развиваются языки программирования. Чтобы язык программирования можно было реально

применять — писать программы на этом языке и исполнять их на компьютере, язык должен быть реализован.

Язык и его реализация

Реализацией языка программирования называют создание комплекса программ, обеспечивающих работу на этом языке. Такой набор программ называется системой программирования. Основу каждой системы программирования составляет транслятор. Это программа, переводящая текст на языке программирования в форму, пригодную для исполнения (на другой язык). Такой формой обычно являются машинные команды, которые могут непосредственно исполняться компьютером. Совокупность машинных команд данного компьютера (процессора) образует его систему команд или машинный язык. Программу, которую обрабатывает транслятор, называют исходной программой, а язык, на котором записывается исходная программа —входным языком этого транслятора.

Компилятор, интерпретатор, конвертор

Компилятор, обрабатывая исходную программу, создает эквивалентную программу на машинном языке, которая называется также объектной программой или объектным кодом. Объектный код, как правило, записывается в файл, но не обязательно представляет собой готовую к исполнению программу. Для программ, состоящих из многих модулей, может образовываться много объектных файлов. Объектные файлы объединяются в исполняемый модуль с помощью специальной программы-компоновщика, которая входит в состав системы программирования. Возможен также вариант, когда модули не объединяются заранее в единую программу, а загружаются в память при выполнении программы по мере необходимости. Интерпретатор, распознавая, как и компилятор, исходную программу, не формирует машинный код в явном виде. Для каждой операции, которая может потребоваться при исполнении исходной программы, в программе-интерпретаторе заранее заготовлена машинная команда или команды. «Узнав» очередную операцию в исходной программе, интерпретатор выполняет соответствующие команды, потом — следующие, и так всю программу. Интерпретатор — это переводчик и исполнитель исходной программы. Различие между интерпретаторами и компиляторами можно проиллюстрировать такой аналогией. Чтобы пользоваться каким-либо англоязычным документом, мы можем поступить по-разному. Можно один раз перевести документ на русский, записать этот перевод и потом пользоваться им сколько угодно раз. Это ситуация, аналогичная компиляции. Заметьте, что после выполнения перевода переводчик (компилятор) больше не нужен. Интерпретация же подобна чтению и переводу

«с листа». Пере-вод не записывают, но всякий раз, когда нужно воспользоваться документом, его переводят заново, каждый раз при этом используя переводчика.

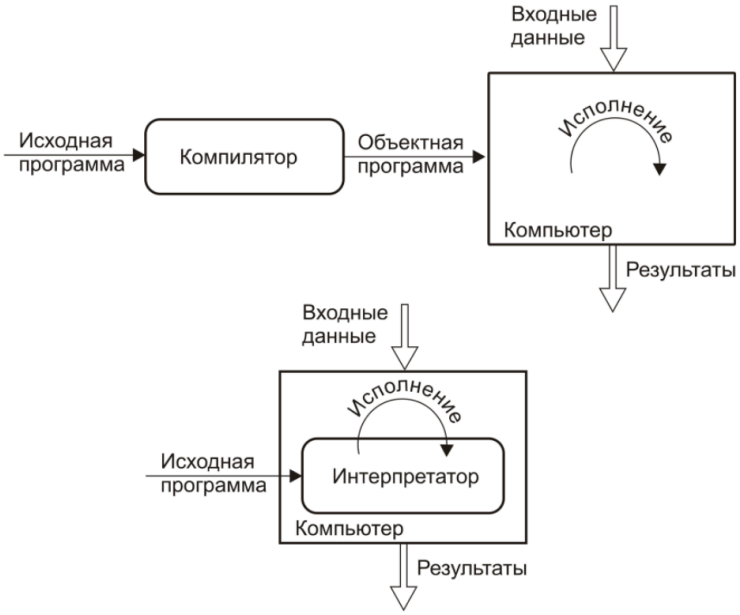

Рис. 1.1.Схема работы компилятора и интерпретатора

Нетрудно понять преимущества и недостатки компиляторов и интерпретаторов. Компилятор обеспечивает получение быстрой программы на машинном языке, время работы которой намного меньше времени, которое будет затрачено на исполнение той же программы интерпретатором. Однако компиляция, являясь отдельным этапом обработки программы, может потребовать заметного времени, снижая оперативность работы. От компилированная программа представляет собой самостоятельный продукт, для использования которого компилятор не требуется. Программа в машинном коде, полученная компиляцией, лучше защищена от внесения несанкционированных искажений, раскрытия лежащих в ее основе алгоритмов.

Интерпретатор исполняет программу медленно, поскольку должен распознавать конструкции исходной программы каждый раз, когда они исполняются. Если какие-то действия расположены в цикле, то распознавание одних и тех же конструкций происходит многократно. Зато интерпретатор может начать исполнение программы сразу же, не затрачивая времени на компиляцию, — получается оперативней. Интерпретатор, как правило, проще компилятора, но его присутствие требуется при каждом запуске программы. Поскольку интерпретатор исполняет программу по ее исходному тексту, программа оказывается незащищенной от постороннего вмешательства.

Но даже при использовании компилятора получающаяся машинная программа работает, как правило, медленнее и занимает в памяти больше места, чем такая же программа, написанная вручную на машинном языке или языке ассемблера квалифицированным программистом. Компилятор использует набор шаблонных приемов для преобразования программы в машинный код, а программист может действовать нешаблонно, отыскивая оптимальные решения. Разработчики тратят немало усилий, стремясь улучшить качество машинного кода, порождаемого компиляторами.

В действительности различие между интерпретаторами и компиляторами может быть не столь явным. Некоторые интерпретаторы выполняют предварительную трансляцию исходной программы в промежуточную форму, удобную для последующей интерпретации. Некоторые компиляторы могут не создавать файла объектного кода. Например, Turbo Pascal может компилировать программу в память, не записывая машинный код в файл, и тут же ее запускать. А поскольку компилирует Turbo Pascal быстро, то такое его поведение вполне соответствует работе интерпретатора.

Существуют трансляторы, переводящие программу не в машинный код, а на другой язык программирования. Такие трансляторы иногда называют конверторами. Например, в качестве первого шага при реализации нового языка часто разрабатывают конвертор этого языка в язык Си. Дело в том, что Си — один из самых распространенных и хорошо стандартизованных языков. Обычно ориентируются на версию ANSI Си — стандарт языка, принятый Американским Национальным Институтом Стандартов (American National Standards Institute, ANSI). Компиляторы ANSI Си есть практически в любой системе.

Кросскомпиляторы (cross-compilers) генерируют код для машины, отличной от той, на которой они работают.

Пошаговые компиляторы(incremental compilers) воспринимают исходный текст программы в виде последовательности задаваемых пользователем шагов (шагом может быть описание, группа описаний, оператор, группа операторов, заголовок процедуры и др.); допускается ввод, модификация и компиляция программы по шагам, а также отладка программ в терминах шагов.

Динамические компиляторы (Just-in-Time — JIT compilers), получившие в последнее время широкое распространение, транслируют промежуточное представление программы в объектный код во время исполнения программы.

Двоичные компиляторы(binary compilers) переводят объектный код одной машины (платформы) в объектный код другой. Такая разновидность компиляторов используется при разработке новых аппаратных архитектур, для переноса больших системных и прикладных программ на новые платформы, в том числе —операционных систем, графических библиотек и др.

Транслятор — это большая и сложная программа. Размер программы-транслятора составляет от нескольких тысяч до сотен тысяч строк исходного кода. Вместе с тем разработка транслятора для не слишком сложного языка — задача вполне посильная для одного человека или небольшого коллектива.

Конструкторский раздел

В описываемой программе несколько основных частей:

- repl - read eval print loop, представляет собой бесконечный цикл, который принимает строку, парусит ее, токенизирует и вычисляет, потом выводит результат пользователю;
- tokenize - переводит распаршенную строку в токены;
- eval - вычисляет строку.

В программе есть следующие токены (конструкции языка Lisp):

```
enum class CellType
{
    Symbol,
    Number,
    List,
    Proc,
    Lambda
};
```

Конструкции языка Lisp хранятся в структуре данных Cell, которая имеет следующий вид:

```
struct Cell
{
    CellType type;

    std::string val;
    std::vector<Cell> list;
    std::function<Cell(const std::vector<Cell> &)> proc;

    std::shared_ptr<Environment> env;

    Cell(CellType type = CellType::Symbol) : type(type) {}
    Cell(CellType type, const std::string &val) : type(type),
val(val) {}
    Cell(std::function<Cell(const std::vector<Cell> &)> proc) :
type(CellType::Proc), proc(proc) {}
};
```

Функция токенизации (непосредственное построение синтаксического дерева):

```
Cell readFrom(std::list<std::string> &tokens)
{
    if (!tokens.empty())
    {
        const std::string token(tokens.front());
        tokens.pop_front();
        if (token == "(")
        {
            Cell cell(CellType::List);
            while (tokens.front() != ")")
            {
                cell.list.push_back(readFrom(tokens));
            }

            tokens.pop_front();
            return cell;
        }
        else
        {
            return atom(std::move(token));
        }
    }
    return Nil;
}
```

Функция вычисления выражения:

```
Cell eval(Cell x, std::shared_ptr<Environment> env)
{
    if (x.type == CellType::Symbol)
    {
        return env->find(x.val)[x.val];
    }
    if (x.type == CellType::Number)
    {
        return x;
    }
    if (x.list.empty())
    {
        return Nil;
    }
    if (x.list[0].type == CellType::Symbol)
    {
        if (x.list[0].val == "quote")  // (quote exp)
        {
            return x.list[1];
        }
        if (x.list[0].val == "if")  // (if test conseq [alt])
        {
```

```
            return eval(eval(x.list[1], env).val == "#f" ?
(x.list.size() < 4 ? Nil : x.list[3])
                                                        :
x.list[2],
                    env);
        }
        if (x.list[0].val == "set!")  // (set! var exp)
        {
            return env->find(x.list[1].val)[x.list[1].val] =
eval(x.list[2], env);
        }
        if (x.list[0].val == "define")  // (define var exp)
        {
            return (*env)[x.list[1].val] = eval(x.list[2], env);
        }
        if (x.list[0].val == "lambda")  // (lambda (var*) exp)
        {
            x.type = CellType::Lambda;
            x.env = env;
            return x;
        }
        if (x.list[0].val == "begin")  // (begin exp*)
        {
            for (size_t i = 1; i < x.list.size() - 1; ++i)
            {
                eval(x.list[i], env);
            }
            return eval(x.list[x.list.size() - 1], env);
        }
    }
    // (proc exp*)
    Cell proc(eval(x.list[0], env));
    std::vector<Cell> exps;
    for (auto exp = x.list.begin() + 1; exp != x.list.end(); +
+exp)
    {
        exps.push_back(eval(*exp, env));
    }

    if (proc.type == CellType::Lambda)
    {
        return eval(/*body*/ proc.list[2],
            std::make_shared<Environment>(/*parms*/
proc.list[1].list, /*args*/ exps, proc.env));
    }
    else if (proc.type == CellType::Proc)
    {
        return proc.proc(exps);
    }

    throw std::runtime_error("Not a function!");
}
```

Технологический раздел

Выбор средств разработки

Выбор целевой платформы

Программный продукт не затачивается под работу на конкретной ОС. Его корректная работа была протестирована на MacOS. Но сама программа не зависит от каких-то конкретных библиотек свойственных только для MacOS, так как написана на чистом C++, а значит портируема на любую существующую ОС.

Выбор языка программирования

Для написания интерпретатора языка Lisp был выбран язык C++. Язык является достаточно развитым и предоставляет все необходимые компоненты для создания такого рода систем. C++ является компилируемым языком программирования и интерпретатор, написанный на нем будет работать достаточно быстро сразу из коробки. Его основные преимущества:

- Скорость работы, программы, написанные на C++, работают быстро, даже с не самыми оптимальными алгоритмами;
- Большая стандартная библиотека, STL развивается уже больше 30 лет и алгоритмы и конструкции, представленные в ней, отлажены и протестированы временем и огромным количеством людей;
- Шаблоны, возможность писать метапрограммы, когда компилятор при разборе текста будет генерировать конструкции на основе описания шаблонов.

Выбор среды разработки и отладки

В качестве среды для разработки использовался текстовый редактор Vim. Данный редактор позволяет настраивать окружение так, как этого хочет пользователь и имеет обширную систему плагинов, разработанных специально для того, чтобы было проще разрабатывать системы. Еще одно из достоинств Vim - его легкость и возможность работы с проектами, у которых обширная кодовая база, так как он не загружает и не индексирует все файлы сам, а использует для этого ctags.

Система контроля версий

В процессе разработки программы использовалась система контроля версий Git. Система контроля версий позволяет вносить в проект атомарные

изменения, направленные на решения каких-либо задач. В случае обнаружения ошибок или изменения требований, несенные изменения можно отменить. Кроме того, с помощью системы контроля версий решается вопрос резервного копирования. Особенности Git:

- Предоставляет широкие возможности для управления изменениями проекта и просмотра истории изменений;
- Данная система контроля версий является децентрализованной, что позволяет иметь несколько независимых резервных копий проекта;
- Поддерживается хостингами репозиториев GitHub, GitLab и BitBucket.

Список использованной литературы

1. Конструирование компиляторов. Учебное пособие. Автор Сергей Свердлов. URL: http://www.uni-vologda.ac.ru/~c3c/CompilersConstruction/Sergey_Sverdlov_Compiler_Construction_(electronic%20version).pdf