П. Б. Кряженков

ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС ИНТЕГРИРОВАННОЙ СРЕДЫ ФУНКЦИОНАЛЬНОГО ПРОГРАММИРОВАНИЯ SFP*

1. ВВЕДЕНИЕ

Функциональный язык программирования, как правило, — язык высокого уровня, предоставляющий богатый набор средств для работы с функциями, рекурсиями. Язык Sisal — один из представителей функциональных языков, ориентированный на сложные вычислительные задачи. Основная синтаксическая единица языка — это функция. В этом главное отличие любого функционального языка от императивного. Операторы языка, такие как циклы, ветвления, присваивания, являются функциями. Несколько меняется программистское мышление, удобнее и понятнее становится писать вычислительные задачи. Математическая завершенность языка гарантирует, что при одних и тех же входных параметрах функция выдаст одинаковые результаты (если не использовать глобально изменяемых переменных). Это позволяет легко отлаживать программы, искать ошибки, проводить оптимизационные преобразования и, что самое примечательное, распараллеливать вычисления уже на уровне функций. Sisal — один из удачных представителей семейства функциональных языков, существующих на данный момент. Программисту предоставляется богатый набор средств для осуществления сложных вычислений, проведения больших научных расчетов. Здесь приведем лишь краткую описательную характеристику, более подробно о Sisal можно узнать, например, из [8]. Как и все современные функциональные языки, он позволяет просто и понятно осуществлять операции с массивами, потоками, отражающими потоки данных. Отличительная особенность — Sisal изначально ориентирован на системы параллельных вычислений. В синтаксисе языка явно присутствуют конструкции, которые будут вычисляться конкурентно. Более того, функциональность и математическая завершенность позволяют проводить эффективный анализ на зависимость по данным и распараллеливать некоторые вычисления, которые предполагалось вести последовательно. Эти преимущества делают

* Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

Sisal все более популярным для осуществления сложных вычислений; появились реализации языка на большинстве известных многопроцессорных архитектур. В силу ориентированности на параллельные вычисления, реализации для персональных компьютеров не создавалось. На маломощных однопроцессорных станциях Sisal — это скорее игрушка, а не инструмент для работы. Реальные преимущества можно почувствовать только на многопроцессорных станциях.

Настоящая статья посвящена некоторым вопросам создания интегрированной среды программирования SFP, которая позволит на маломощных персональных компьютерах выполнять такие ресурсомалоемкие задачи, как набор исходных текстов, выполнение тестовых запусков, отладка программы, эмулирование многопоточности и многозадачности, а затем по сети передавать код на многопроцессорную аппаратную платформу для полного счета. Таким образом ресурсы мощной станции будут использоваться в основном для счета, т.е. ресурсоёмких задач. Так достигается эффективность и производительность вычислительных систем.

Данная статья имеет следующую структуру. Разд. 1 — введение, содержит описание поставленной задачи, обосновывает ее актуальность. Далее идет разд. 2, описание пользовательского интерфейса среды SFP. В разд. 3 описывается реализация редактора системы, какие проблемы возникали и как были решены. Разд. 4 кратко описывает процесс трансляции программы на языке Sisal из текстового в промежуточное представление. Описание разбито на три части — лексический, синтаксический и семантический анализы. И разд. 5 — заключение, результаты проделанной работы и планы на будущее.

2. ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС СРЕДЫ SFP

Для запуска интегрированной среды достаточно запустить исполняемый файл среды. Система инициализируется из конфигурационного файла, считывая оттуда параметры среды, настраиваемые на конкретного пользователя, такие как шрифт текстового редактора, цвета, используемые при подсветке синтаксических единиц языка и т.п. Первое, что видит пользователь, — главное окно среды с верхним меню, панелью инструментов (tool bar). Внизу находится строка состояния (status bar), где отображаются динамические подсказки (кратко описывающие действия, которые можно произвести, “кликнув” в данном месте на мышку), состояние клавиш caps/num/scroll lock. Здесь не открыто ни одного файла для редактирования,

потому некоторые кнопки панели инструментов недоступны (они затемнены серым цветом, доступные кнопки панели инструментов выведены в полном цвете), такие как сохранение файла, кнопки работы с буфером обмена, печать на принтере. Меню верхнего уровня приложения в данном случае выглядит следующим образом: File (работа с файлами), View (вид), Help (помощь). Подчеркнутые буквы служат для обозначения горячих клавиш, которые в сочетании с нажатием клавиши Alt вызовут активацию соответствующего пункта меню.

В меню File доступны следующие команды: New (создание нового текстового файла), Open (открытие существующего файла), Print Setup…(установки принтера), 1 2 3 4 (четыре пункта на открытие последних 4 редактированных файлов в редакторе), Exit (выход). Команда New создает новый текстовый файл с именем EditorNN (NN — соответствующий порядковый номер) с пустым содержимым и открывает для него окно редактирования. Можно набирать в открывшемся окне текст. Команда Open выводит на экран стандартный блок диалога File/Open для выбора файла, который предполагается редактировать. После того как пользователь выбрал файл, открывается окно редактирования, в котором текст и выводится. Print Setup представляет стандартный сервис библиотеки MFC для настройки параметров принтера. Следующие четыре пункта показывают, какие последние четыре файла пользователь редактировал в среде. По нажатию на имя соответствующего файла он открывается на редактирование, как будто был выбран в процессе диалога File/Open команды меню Open. Пункт меню Exit — выход из системы. Пока не открыто ни одного файла на редактирование, выполняется простой выход из системы, освобождаются системные ресурсы, занятые приложением, и программа завершает свое исполнение.

Меню View содержит следующие пункты: Toolbar (панель инструментов) и Status bar (строка состояния), которые могут быть в состоянии checked (выбрано) или unchecked (не выбрано). Состояние checked означает, что соответствующий элемент пользовательского интерфейса (панель инструментов, строка состояния) будет отображен в окне редактора, состояние unchecked — будет скрыт.

Меню Help на данном этапе состоит из одного пункта — About Editor...(описание...). Выбор данного пункта меню выводит на экран блок диалога с краткой информацией о системе: версия, разработчик, дата создания.

Рассмотрим более подробно другие элементы пользовательского интерфейса: панель инструментов и строку состояния. Недоступные кнопки

панели инструментов (сохранение файла, работа с буфером обмена Windows) не имеют аналогов в командах меню в данный момент. Активные кнопки (New, Open, About) дублируют команды, вызываемые активацией соответствующих пунктов верхнего меню (File/New, File/Open, Help/About Editor). Панель инструментов является плавающей. Это означает, что пользователь может перемещать ее в пределах главного окна среды, “приклеивать” (dock) к краям, а также помещать в “неприклеенном” состоянии в любом месте. При наведении курсора мышки на кнопку из панели инструментов высвечивается быстрая подсказка (tooltip) около курсора и отображается развернутая подсказка в строке состояния, внизу главного окна. Строка состояния отвечает за вывод соответствующего описания команд панели инструментов, некоторой другой справочной информации, а также за отображение состояния клавиш caps/num/scroll lock.

Несколько иным выглядит внешний вид среды (рис. 1) в рабочем состоянии, если открыты файлы на редактирование. Меню верхнего уровня кардинально изменяется, в главном окне среды отображаются окна редактирования файлов (одно или несколько), также может присутствовать панель инструментов и строка состояния главного окна. У каждого окна редактирования есть своя строка состояния, которая содержит информацию о редактируемом в нем тексте: положение курсора (номер строки/колонки), режим ввода (вставка/переписывание символов). Также внутри окон редактирования доступно контекстное меню.

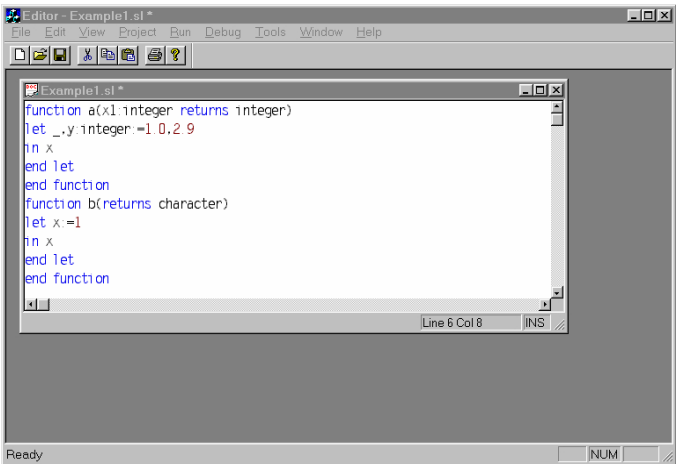

Рис. 1

Меню верхнего уровня включает: File (работа с файлами, выход), Edit (команды редактирования и поиска текста), View (вид), Project (команды управления проектом), Run (запуск), Debug (отладка), Tools (инструменты, настройки), Window (управление окнами редактирования), Help (помощь). Меню View и Help аналогичны меню, описанным в предыдущем пункте. Некоторые меню в системе не реализованы (Project, Debug), они присутствуют в расчете на будущую доработку. Опишем каждое меню более подробно. Некоторые пункты в соответствующих меню также не реализованы (они подсвечены серым цветом и не вызывают никакого действия). Они также присутствуют в системе на будущее.

Меню File состоит из тех же пунктов, которые были описаны в прошлом разделе (New, Open, Print Setup, Exit), с добавлением некоторого количества новых: Close (закрыть), Save (сохранить), Save As…(сохранить под другим именем), Print (печать), Print preview (предварительный просмотр перед печатью). Команда Close вызывает закрытие текущего окна редактирования с проверкой, если файл не был сохранен, то выдается запрос на сохранение файла. Save сохраняет текущий редактируемый файл. Save As… сохраняет текущий редактируемый файл под другим именем, выдается стандартный блок диалога File/Save, где пользователь может выбрать имя записываемого файла, после чего производится запись. Print и Print preview предоставляют интерфейс библиотеки MFC по умолчанию в архитектуре документ/представление.

Меню Edit состоит из следующих подпунктов: Undo (отмена), Cut (вырезать), Copy (копировать), Paste (вставить), Find (найти). Команда Undo отменяет последнее действие, произведенное пользователем в окне редактирования (вставка/удаление символов или блоков текста с клавиатуры или из буфера обмена Windows). Следующие три пункта предназначены для работы с буфером обмена Windows. Cut удаляет выделенный текст из окна редактирования, одновременно копируя его в буфер обмена. Copy просто копирует выделенный фрагмент текста в буфер обмена. Paste вставляет блок текста из буфера обмена в текущей позиции курсора в редактируемом тексте. Команда Find предназначена для поиска фрагмента текста в текущем окне. По этому пункту меню выдается диалоговое окно, где пользователь может задать параметры поиска, такие как сам фрагмент, учет верхнего/нижнего регистра символов, откуда и где искать (сначала текста, от позиции курсора, с конца текста), объем просматриваемого текста (весь или только выделенный), направление поиска (вперед, назад). Результат диалога — новый поиск (Find) или продолжение старого (Find next).

Меню Run содержит пока только один активный пункт — Check syntax (проверить синтаксис). Результат его применения — текущий редактируемый текст пропускается через синтаксический анализатор на предмет выявления синтаксических или семантических ошибок, которые выводятся в соответствующем окне. В случае отсутствия ошибок строится дерево грамматического разбора, которое выдается пользователю в блоке диалога.

Меню Tools состоит из следующих пунктов: IDE Options… (параметры интегрированной среды) и Reread “editor.ini” (перечитать файл конфигурации editor.ini).

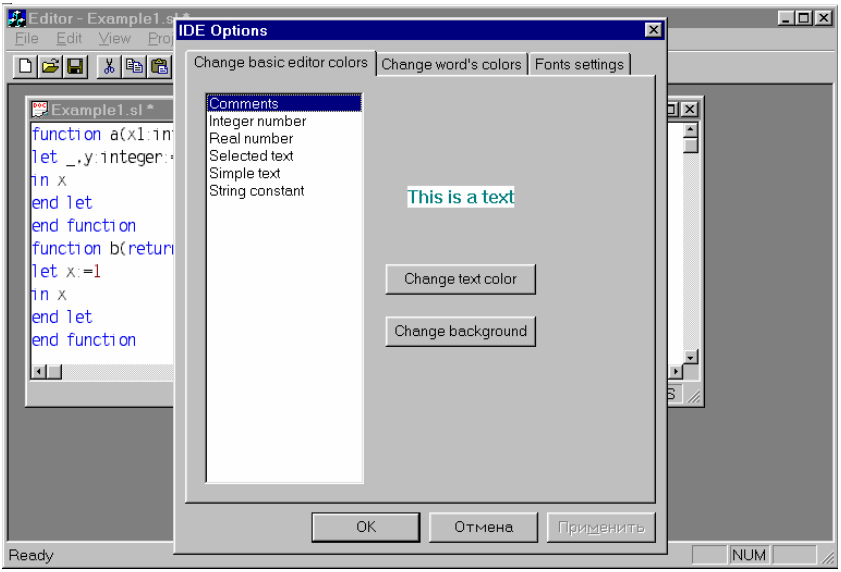

Рис. 2

IDE Options (рис. 2) служит для изменения параметров интегрированной среды программирования. По этой команде на экран выдается блок диалога, состоящий из трех подблоков: Change basic editor color (изменение цветов для синтаксических единиц языка и основных цветов редактора), Change word’s colors (изменение цветов для ключевых слов, рис. 3), Font settings (параметры шрифта редактора). Change basic editor color — в этом блоке диалога можно выбрать, для какого элемента будем менять цвет (комментарии, целые, вещественные или строковые константы, а также выделенный и простой текст), и изменить основной и фоновый цвет.

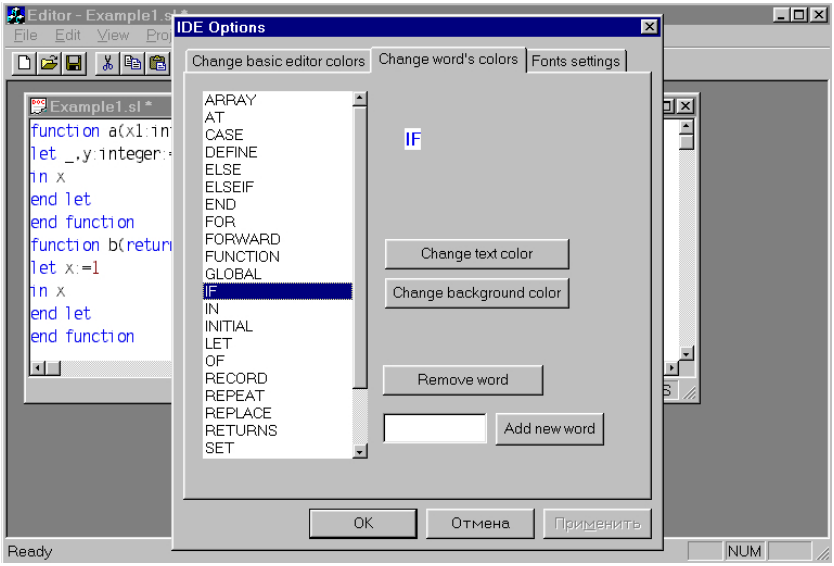

Рис. 3

Change word's colors — здесь можно выбрать одно или несколько ключевых слов и изменить для них цвет фона и тона. Также доступны функции по редактированию набора ключевых слов: добавление и удаление. Font settings — изменение шрифта, используемого в редакторе. Все изменения, сделанные по IDE Options, в случае выхода по клавише "OK" начинают действовать немедленно и сохраняются в конфигурационном файле "editor.ini" (обзор части системы, отвечающей за работу с конфигурационным файлом см. позднее). Reread "editor.ini" перечитывает конфигурационный файл, и все новые настройки начинают действовать немедленно. Данная команда может использоваться для глобального изменения всех настроек без перезапуска системы.

Меню Window отвечает за управления окнами редактирования. Состоит из следующих пунктов: Cascade (каскадом), Tile (мозаикой), 1 2 ... перечислены открытые файлы, в которые можно переключиться для редактирования. Команда Cascade располагает окна редактирования одно над другим, каждое следующее окно чуть смещается от предыдущего по диагонали. Tile располагает окна редактирования в виде мозаики так, что все окна стано-

вятся видны на экране. 1 2 … осуществляет активизацию соответствующего окна и переключает на него фокус ввода. Данные пункты меню позволяют более удобно редактировать текст, если он разбит на несколько файлов.

Панель инструментов состоит из тех же кнопок, только в данном состоянии они все активны. Нажатие на какую-либо кнопку панели инструментов вызывает исполнение определенной команды верхнего меню. Это сделано для удобства пользователя. New соответствует File/New, Open — File/Open, Save — File/Save, Cut — Edit/Cut, Copy — Edit/Copy, Paste — Edit/Paste, Print — File/Print, About — Help/About Editor…

Каждое окно редактирования обладает своей строкой состояния, где отображается информация о текущем редактируемом тексте, а также контекстным меню, которое может быть вызвано нажатием правой клавиши мыши. Контекстное меню содержит команды для ускорения работы пользователя: Copy (копировать), Cut (вырезать), Delete (удалить), Paste (вставить). Delete удаляет выделенный фрагмент текста (как будто нажата клавиша Del), остальные команды работают с буфером обмена Windows: Copy копирует выделенный фрагмент текста в буфер обмена, Cut удаляет из текста и копирует в буфер обмена выделенный фрагмент текста, Paste вставляет блок текста из буфера обмена в текущую позицию курсора.

Интегрированная среда является Windows приложением, а поэтому в нем активно используется мышь. В данном разделе предлагается краткое описание и особенности реализации некоторых функций, которые можно выполнить с помощью мыши. Сюда входят стандартные Windows-функции, как выбор пунктов меню с помощью щелчка на соответствующем названии левой кнопкой мыши, вызов контекстного меню окон редактирования по щелчку правой кнопки мыши, сворачивание/разворачивание и закрытие окон, имеющих системное меню. Используя полосу прокрутки, можно быстро и удобно перемещаться по тексту. Особенно просто, понятно и элегантно можно выделять фрагменты текста с помощью мыши и перемещаться к нужному символу, видимому на экране (для последнего достаточно просто щелкнуть левой кнопкой на символе). В случае выделения большого фрагмента, текст в редакторе автоматически скроллируется в нужном направлении для продолжения выделения.

3. РЕДАКТОР. ОПИСАНИЕ И РЕАЛИЗАЦИЯ

Архитектура документ/представление (document/view) предназначена для упрощения процесса разработки приложения для Windows с помощью

библиотеки MFC. В основе этой архитектуры лежат три глобальных понятия — фрейм, документ и представление. Под документом понимаются те данные, с которыми работает приложение (текст, картинка, ...). Отображение этих данных на экране осуществляется во фрейме документа. В библиотеке для фреймов предусмотрены специальные классы окон — представления, которые отображают данные документа и управляют взаимодействием пользователя с ними. Таким образом, способ хранения данных в памяти или на диске никоим образом не влияет на их внешнее представление пользователю.

В данном редакторе использована архитектура документ/представление. Объект *документ* хранит текст в виде списка строк в памяти, отвечает за операции ввода/вывода (запись и чтение файла). Объект *представление* отвечает за отображение текста на экране пользователя, он хранит информацию, отвечающую за отображение текста, такую как номер текущей строки, колонки курсора, текущий используемый шрифт и т.п. Также на объект представление ложится вся первичная обработка действий, запрошенных пользователем, такие как вставка/удаление текста, перемещения курсора, выделение мышью, работа с буфером обмена Windows.

Редактор интегрированной среды обладает возможностью подсветки текста. Каждой из следующих конструкций можно назначить отдельный цвет: простому тексту, константам (целым, вещественным и строковым) и комментариям. Также цвет можно задать отдельно каждому ключевому слову. Под выражением *задать цвет* понимается задать, каким цветом тона/фона будет изображаться данная конструкция на экране монитора. Цветовые настройки доступны из подблоков диалога в меню Tools/IDE Options. Все изменения в параметрах вступают в силу немедленно.

Реализация подсветки в редакторе достаточно проста: текст раскрашивается построчно. Внутри одной строки сначала выделяются слова, если слово ключевое, то оно закрашивается соответствующим цветом. На следующем шаге проверяется, распознана ли одна из следующих синтаксических конструкций: целая, вещественная или строковая константа. Если нет, то текст изображается как простой текст. После этих шагов, если в строке есть комментарии, то закрашиваются комментарии. И последний шаг, если в строке есть выделенный фрагмент текста, то он окрашивается в нужный цвет. Такой алгоритм отрисовки позволяет перерисовывать автономно только одну строку текста, что можно использовать для оптимизации работы редактора.

Все настройки интегрированной среды (шрифты, параметры цветов) хранятся в конфигурационном файле “editor.ini”. Формат хранения пара-

метров следующий: имя_параметра = значение_параметра. Для текстовых параметров (например ключевое слово) параметры хранятся в явном виде (имя_параметра = текстовое_значение_параметра). Если параметр не является текстовым, то он преобразуется к текстовому виду таким образом, чтобы его можно было восстановить по текстовому представлению. Способ преобразования для каждого параметра свой. Например: C_INTEGER_BK = 255,255,255 означает, что rgb-компоненты цвета фона для целых констант есть соответственно 255,255,255. Регистр символов, используемых при записи параметров, не учитывается, ведущие пробелы игнорируются. В случае переопределения значения параметра (левая часть встречается более одного раза в конфигурационном файле) используется последнее определение.

Опишем кратко реализацию поддержки конфигураций в системе. За все конфигурационные параметры отвечает класс CInitParameters, единственный экземпляр которого создается в объекте CEditorApp (потомок класса CWinApp, стандартный объект приложения в библиотеке классов MFC). В архитектуре документ/представление доступ к экземпляру класса CEditorApp возможен в любом месте. Поэтому экземпляр класса CInitParameters "виден" в любом месте программы и все инициализационные параметры доступны. Данный подход позволяет изменять любые конфигурационные параметры "на лету".

Кратко опишем параметры конфигурационного файла. C_<элемент_подсветки>_Text/Bk — параметр, задающий цвет тона/фона для элемента подсветки. Элемент_подсветки может быть одним из следующих: integer (целая константа), real (вещественная константа), string (строковая константа), comment (комментарий), selected (выделенный текст), <пусто> (обычный текст). C_Word_<ключевое_слово>_Text/Bk — параметр, задающий цвет тона/фона для соответствующего ключевого слова. Значениями описанных выше параметров могут быть тройки десятичных чисел, разделенных запятой: r, g, b, где r, g, b — соответственно значения красной, зеленой и синей компоненты цвета. EditorFont = NNNN… — параметр, определяющий текущий шрифт редактора. NNN… — двоичное представление объекта LOGFONT. Параметр EditorFont вручную менять не рекомендуется, это может привести к непредсказуемым последствиям. Остальные параметры можно изменять в соответствии с приведенными правилами, и хотя для каждого есть визуальная настройка, может понадобиться доводка параметров вручную.

Очень просто производится настройка параметров на конкретного пользователя. После настройки параметров среды надо сохранить конфигураци-

онный файл, и позже в любой момент положить его в каталог системы и дать команду Reread “editor.ini” в меню Tools. Все изменения вступят в силу немедленно**.**

4. ТРАНСЛЯЦИЯ И ПОСТРОЕНИЕ ПРОМЕЖУТОЧНОГО ПРЕДСТАВЛЕНИЯ

Как правило, процесс трансляции состоит из трех основных частей: лексический анализатор, синтаксический анализатор и анализатор семантики. На этапе лексического анализа производится так называемая лексическая свертка программы. Входная программа рассматривается как одна длинная строка, она анализируется посимвольно, и выделяются отдельные лексемы. Представленная таким образом программа подается на вход синтаксического анализатора, задача которого проверить, удовлетворяет ли данная входная строка синтаксису языка. Как правило, одновременно строится промежуточное представление программы для последующего семантического анализа либо семантический анализ может проводиться параллельно с синтаксическим.

Лексический анализ традиционно занимает много машинного времени по сравнению с остальными этапами процесса трансляции. Несмотря на простоту анализа и понятность работы, входная строка (которая может быть достаточно длинной) подвергается посимвольной обработке.

В данной работе применен алгоритм построения непрямого лексического анализатора. Выбранное подмножество синтаксиса языка имеет достаточно простое строение для лексического распознавания. Лексический анализатор интегрирован с синтаксическим анализатором, сгенерированным YACC, потому имеет некоторые характерные особенности, которые будут описаны ниже. Здесь упомянем только то, что при распознавании очередной лексемы возвращается ее код, а само значение лексемы (число для вещественных и целых чисел, строка для идентификатора и т.п.) помещается в специальную переменную. Еще хочется отметить, что специально для YACC существует генератор лексических анализаторов LEXX, однако при разработке среды он не был использован. Как уже отмечалось выше, подмножество синтаксиса языка имеет достаточно простое строение, и потому прямая реализация лексического анализатора эффективнее.

Алгоритм работы реализованного непрямого лексического анализатора следующий:

1. Сначала пытаемся распознать специальные символы, такие как скобки, точки с запятой, запятые, знаки операций, присваивания. В случае успеха возвращается код соответствующего специального символа, никакого специального значения в служебные переменные не помещается.
2. Если это не специальный символ, то значит, либо константа (числовая или строковая), либо идентификатор. Все эти синтаксические конструкции легко определяются по первому символу: если цифра, то это числовая константа; если это буква — идентификатор, если “ — строковая константа; иначе — ошибка. Следующие шаги описывают чуть более подробно, как распознается каждая из этих конструкций.
   2.1. Числовая константа: целая или вещественная. Сначала пытаемся распознать вещественную константу, если удалось, то возвращаем сигнал, что распознана вещественная константа, а значение помещаем в служебную переменную. Если неуспех, то пытаемся распознать целую константу (в этом обязательно будет успех, т.к. первую цифру уже распознали). Возвращаем код, что распознана целая константа, а значение помещаем в служебную переменную.
   2.2. Идентификатор распознается как набор букв, цифр, знака “_”, начинающийся с буквы или _. После распознавания идентификатор заносится (если необходимо) в глобальную таблицу идентификаторов, получается код идентификатора. Результат работы анализатора в этом случае — возвращается сигнал, что распознан идентификатор, а его код помещается в служебную переменную.
   2.3. Строковая константа — все, что встречается во входной строке до следующего знака “, считается строкой. После распознавания строковая константа помещается в таблицу, получается ее код. Возвращается сигнал, что распознана строковая константа, а ее код помещается в служебную переменную.
   2.4. Ошибка. Выдается сигнал ошибки, предпринимаются специальные действия реакции на ошибку (они будут описаны ниже, при рассмотрении синтаксического анализатора).

Как уже говорилось выше, в данной работе при построении синтаксического анализатора использовался инструмент YACC (Yet Another Compiler to Compiler), предназначенный вообще для создания программ, проверяющих корректность входных данных. Полное описание можно посмотреть в Интернете, например [8]. YACC принимает на вход набор грамматических правил, формально описывающих синтаксис входного языка, и генерирует

восходящий LALR(1) синтаксический анализатор в виде подпрограммы на C yyparse(). Построенный синтаксический анализатор обращается к лексическому анализатору, когда ему требуется очередная лексема, который должен написать сам пользователь. Нужная информация по технологическим особенностям YACC будет даваться по ходу изложения того, как построен синтаксический анализатор. Здесь же хотелось бы привести некоторые рассуждения о преимуществах и недостатках построения парсеров с помощью YACC и вручную, а также указать на некоторые проблемы, возникшие при реализации, и пути их решения.

Преимущества заключаются в следующем: спецификацию входного языка достаточно задать в виде набора грамматических правил, что вполне понятно и наглядно; программисту следует мыслить в терминах грамматик, а не в терминах конструкций языка программирования, что значительно снижает вероятность ошибки и позволяет создавать поддержку более крупных синтаксических конструкций. К недостаткам стоит отнести то, что с помощью YACC можно создавать синтаксические анализаторы только для LALR(1) языков (и небольших модификаций). Поддержка обработки ошибок хотя и существует в YACC, однако с гибкостью и легкостью прямых методов программирования здесь сравнивать сложно. Анализаторы, написанные с помощью YACC, достаточно легко расширять поддержкой дополнительных синтаксических конструкций, что при прямых методах делается иногда с трудом.

К проблемам, которые встретились при реализации парсера с помощью YACC, можно отнести некоторую громоздкость получаемых текстов (даже при неполной реализации синтаксиса языка!), отсутствие каких-либо визуальных средств конструирования грамматик. Решение состоит в разбиении на отдельные файлы части текста грамматик, которые отвечают за различные синтаксические конструкции, а для сборки написан небольшой скрипт, собирающий все в один файл. Реализованное подмножество языка Sisal в виде набора грамматических правил, которые подаются на вход YACC, приведено в приложении.

Семантический анализ — следующий этап в процессе трансляции после синтаксического анализа. Подробно об этом можно прочитать в специальной литературе, посвященной построению трансляторов, интерпретаторов и компиляторов, например, [1, 4–6].

Опишем реализацию семантического анализа в данной работе. Проверка семантики языка Sisal осуществляется параллельно синтаксическому анализу, при построении дерева разбора. После того как разобрана очеред-

ная конструкция, построено поддерево, для него вычисляются атрибуты, проверяется корректность использования имен, совпадение типов и, если требуется, автоматическое преобразование типов (например, целый тип расширяется до вещественного; если тип явно не задан пользователем, то задается на данном этапе).

Для арифметических выражений вычисляется атрибут — тип выражения на основе наследуемых поддеревьев. Например, для операции сложения:

a_expr: a_expr + a_expr;

атрибут-тип для конечного выражения вычисляется из типов операндов (если real и integer, то получившееся выражение будет иметь тип real).

К семантическим проверкам, применяемым к арифметическим выражениям, следует отнести проверку корректности использования имен переменных, имен функций (чтобы они были видны в данном контексте), соответствия типов арифметическим (нельзя использовать символьные и логические типы), а также возможность расширения типа выходного выражения (если оно явно задано пользователем). Обязательное требование для вызовов функций — чтобы они возвращали только одно значение арифметического типа.

Рассмотрим семантику оператора цикла for_initial, формальное описание приведено выше. Цикл выполняется, пока истинно логическое условие в while-части оператора. Проверяется, возможно ли использовать переменные и функции в редукциях, операторах присваивания и логических выражениях, а также вычисляются типы возвращаемых оператором значений. Для определения функций также вычисляются типы возвращаемых значений, проверяется, чтобы оператор тела возвращал соответствующие типы, использование имен переменных, аргументов.

Семантический анализ завершает этап процесса трансляции. В случае отсутствия ошибок пользователю выдается сообщение об успехе, иначе неправильные конструкции фиксируются, и в интерактивном режиме предлагается их исправить.

5. ЗАКЛЮЧЕНИЕ

В данной статье рассмотрены вопросы создания компонент интегрированной среды SFP. Кратко описаны внешняя спецификация, пользовательский интерфейс и аспекты реализации следующих компонент интегриро-

ванной среды: редактор, синтаксический анализатор с возможностью интерактивного исправления ошибок. Разработка компонентов интегрированной среды продолжается, в скором времени планируется добавить в список доступных интерпретатор и отладчик с визуальной навигацией по коду. В отладчике должна быть возможность пошагового выполнения программы блоками (например, функциями) до определенной точки останова или до конца программы. Также рассматривается возможность добавления в интегрированную среду преобразователя промежуточного представления для более эффективного исполнения.

СПИСОК ЛИТЕРАТУРЫ

1. **Касьянов В. Н., Поттосин И. В.** Методы построения трансляторов / Отв. ред. А.П. Ершов. — Новосибирск: Наука, 1986. — 344 с.
2. **International** Workshop, selected papers. — Implementation of functional languages. — 1995.
3. **Бирюкова Ю. В.** Sisal 90. Руководство пользователя. — Новосибирск, 2000.
4. **Варсанофьев Д. В., Дымченко А. Г.** Основы компиляции. — Москва. 1991.
5. **Брежнев А. М.** Системное программное обеспечение, трансляторы. — Северодонецк, 1995.
6. **Серебряков В. А.** Лекции по конструированию компиляторов. — Москва, 1993.
7. **Информация** по YACC и созданию трансляторов: http://yacc.chat.ru.
8. **Касьянов В. Н., Бирюкова Ю. В., Евстигнеев В. А.** Функциональный язык Sisal 3.0 // Поддержка супервычислений и интернет-ориентированные технологии. — Новосибирск, 2001. — С. 54–67.

П р и л о ж е н и е

СПЕЦИФИКАЦИЯ ПОДМНОЖЕСТВА ЯЗЫКА SISAL В ВИДЕ НАБОРА ГРАММАТИЧЕСКИХ ПРАВИЛ, ПОНЯТНЫХ YACC

```
%start list

%token NUMBER REAL POWER_OP IDENTIFIER KEYWORD_LET KEY-
        WORD_IN KEYWORD_END
        ASSIGN_OP/*:=*/ KEYWORD_TRUE KEYWORD_FALSE
        NOT_EQUAL LESS_OR_EQUAL GREATER_OR_EQUAL
        KEYWORD_IF KEYWORD_THEN KEYWORD_ELSEIF KEY-
        WORD_ELSE
        KEYWORD_FOR KEYWORD_INITIAL KEYWORD_WHILE KEY-
        WORD_REPEAT
        KEYWORD_RETURNS KEYWORD_OF KEYWORD_ARRAY KEY-
        WORD_SUM KEYWORD_PRODUCT
        KEYWORD_VALUE KEYWORD_LEAST KEYWORD_GREATEST
        KEYWORD_FUNCTION KEYWORD_INTEGER KEY-
        WORD_CHARACTER


%left '-' '+'
%left '*' '/'
%left UNARYMINUS UNARYPLUS /*унарные плюс и минус*/
%right POWER_OP /*операция возведения в степень*/
%left '>' '<' '=' NOT_EQUAL LESS_OR_EQUAL GREATER_OR_EQUAL
%left '|' '&'
%right '~'

%%

list:   /*nothing*/     {;}
        | function_list  | list error;

operator: let_operator | if_operator | for_initial_operator | error;
```

```
/*************************************************************
        описываем функцию
*/
function_list:    function | function_list function;

function:  KEYWORD_FUNCTION IDENTIFIER '(' arguments_list KEY-
      WORD_RETURNS return_type_list ')' operator KEYWORD_END
      KEYWORD_FUNCTION;
/*список аргументов функции*/
arguments_list: /*nothing*/ | arguments_type | arguments_list ';' arguments_type;
/*список переменных(аргументов) одного типа*/
arguments_type:      identifier_list ':' type_declaration;
/*имя типа(декларация имени типа)*/
type_declaration: type_name;
/*имя типа*/
type_name:      reserved_type;
reserved_type:  KEYWORD_INTEGER   | KEYWORD_CHARACTER;
identifier_list: IDENTIFIER | identifier_list ',' IDENTIFIER;
/*список имен типов, возвращаемых функцией*/
return_type_list: type_declaration | return_type_list ',' type_declaration;


 /*************************************************************
  Секция описания операторов
*/


for_initial_operator: KEYWORD_FOR KEYWORD_INITIAL assignment_list
      KEYWORD_WHILE boolean_expr KEYWORD_REPEAT assign-
      ment_list KEYWORD_RETURNS reduction_list KEYWORD_END
      KEYWORD_FOR
      | error;
reduction_list:  reduction;
reduction: reduction_word KEYWORD_OF expr;
reduction_word:     KEYWORD_VALUE | KEYWORD_ARRAY | KEY-
      WORD_SUM | KEYWORD_PRODUCT | KEYWORD_LEAST | KEY-
      WORD_GREATEST;
```

```
let_operator:    KEYWORD_LET assignment_list
            KEYWORD_IN expr_list KEYWORD_END KEYWORD_LET
            | error;
if_operator :    KEYWORD_IF boolean_expr KEYWORD_THEN expr_list
        KEYWORD_ELSE expr_list KEYWORD_END KEYWORD_IF
        | KEYWORD_IF boolean_expr KEYWORD_THEN expr_list elseif_list
        KEYWORD_ELSE expr_list KEYWORD_END KEYWORD_IF
        | error;
elseif_list: KEYWORD_ELSEIF boolean_expr KEYWORD_THEN expr_list
                | KEYWORD_ELSEIF boolean_expr KEYWORD_THEN
        expr_list elseif_list;


/*****************************************************************
            здесь описываются присваивания
*/
assignment_list: assignment | assignment_list ';' assignment;
expr_list:  expr | expr_list ',' expr;
expr:   number_expr | boolean_expr;

/*одиночное присваивание*/
assignment:      variable_declaration_list ASSIGN_OP expr_list | error;
/*объявление списка переменных(типизированных или нет)*/
variable_declaration_list: variable_declaration | variable_declaration_list ',' vari-
        able_declaration;
/*объявление типизированной или нетипизированной переменной*/
variable_declaration: IDENTIFIER ':' type_declaration | IDENTIFIER;

/*****************************************************************
          вызов функции — имя и параметры
*/
function_call:   IDENTIFIER '(' expr_list ')' | IDENTIFIER '(' ')';


/*****************************************************************
          Числовые выражения*/
number_expr: NUMBER_CONSTANT
                | function_call
                | IDENTIFIER
```

```
                | operator
                | '-' number_expr %prec UNARYMINUS
                | '+' number_expr %prec UNARYPLUS
                | number_expr POWER_OP number_expr
                | number_expr '+' number_expr
                | number_expr '-' number_expr
                | number_expr '*' number_expr
                | number_expr '/' number_expr
                | '(' number_expr ')';
NUMBER_CONSTANT: NUMBER | REAL;

/************************************************************
        булевское выражение*/
boolean_expr:   BOOLEAN_CONSTANT
                | function_call
                | number_expr '>' number_expr
                | number_expr '<' number_expr
                | number_expr '=' number_expr
                | number_expr NOT_EQUAL number_expr
                | number_expr LESS_OR_EQUAL number_expr
                | number_expr GREATER_OR_EQUAL number_expr
                | '~' boolean_expr
                | boolean_expr '|' boolean_expr
                | boolean_expr '&' boolean_expr
                | '(' boolean_expr ')';
BOOLEAN_CONSTANT: KEYWORD_TRUE | KEYWORD_FALSE;
%%
```