Руководство Oakwood для разработчиков компилятора Oberon-2

Область применения

Этот документ является дополнением к отчету ETH Oberon-2 и содержит пояснения, расширения, рекомендации по реализации и базовое определение библиотеки.

Назначение

Документирование обсуждений, состоявшихся на конференции Oakwood в Кройдоне в 1993 году, и предоставление практических рекомендаций для авторов компиляторов. Цель - выработать общий подход к реализации компиляторов Oberon-2 на различных платформах и поощрять согласованность там, где это возможно.

Авторы

См. Приложение 2

Редакция: 1A Первый выпуск
Под редакцией:Брайан Кирк, Робинсон Ассошиэйтс

Посвящение

Этот документ посвящен памяти Ника Уолша.

Его здравых советов и тонкого остроумия в

сочетании с интеллектуальной ясностью будет

очень не хватать друзьям, коллегам и

студентам.

Предисловие

Язык Oberon вместе с системой Oberon был разработан и внедрен профессором Никлаусом Виртом и профессором Иргом Гуткнехтом в ETH Zrich с 1985 по 1987 год. С тех пор она интенсивно используется для преподавания, а также для проектов в ETH и других странах. После ряда незначительных изменений, которые также привели к появлению Oberon-2, язык наконец стал стабильным и зрелым. В настоящее время он доступен практически на всех современных платформах. Все эти реализации поддерживают один и тот же язык и даже одни и те же интерфейсы для файлов, win- dows и других ресурсов операционной системы. Таким образом, можно говорить о стандарте де-факто.

Такова была ситуация, когда группа из примерно 30 разработчиков и поставщиков компиляторов собралась в отеле Oakwood в Кройдоне в июне 1993 года, чтобы договориться об общем наборе языковых функций и библиотечных модулей, которые должны быть предоставлены каждой системой Oberon-2.

Эта группа работала очень эффективно, избегая бюрократии и длительных совещаний. В течение нескольких месяцев они выпустили документ, который включает в себя результаты встречи в Дубовом лесу и устанавливает набор рекомендаций для разработчиков компиляторов.

Помимо некоторых уточнений к протоколу языка и скромного набора возможных расширений, центральная часть этого документа дает подсказки разработчикам компиляторов и определяет базовую библиотеку модулей, которая должна поставляться с каждой реализацией Oberon-2.

Я надеюсь, что будущие разработчики систем Oberon-2 будут придерживаться этих рекомендаций во благо единообразия и переносимости. Oberon-2 будет успешным только в том случае, если не повторит ошибок реализаций Modula-2, где отсутствие согласия между ранними разработчиками ком-пиляторов привело к несовместимости и слишком долгому процессу стандартизации.

Особую благодарность я выражаю Брайану Кирку и Юэну Хиллу, которые выступили в роли организаторов встречи в Оуквуде, а затем взяли на себя трудную задачу собрать и свести все замечания, предложения и пожелания в единый и достаточно короткий документ. Спасибо также всем людям, перечисленным в конце этого документа, которые внесли свой вклад в духе сотрудничества.

Ханспетер

Мёссенбёк ETH,

Цюрих Ноябрь, 1993

Заметки редактора

Представляем вашему вниманию первый выпуск Руководства Oakwood. Он основан на проекте, распространенном по электронной почте, который был исправлен на основе ваших отзывов и рассмотрен профессором Мёссен-бёком и Йозефом Темплом из ETH. Я понимаю, что, возможно, все авторы могут быть разочарованы! Причина проста: было выдвинуто много идей, и я постарался включить только те, которые обсуждались на встрече в Оквуде или имеют действительно сильную поддержку. Пункты в скобках << как здесь >> выделяют темы, которые требуют дальнейшего уточнения. Любые ошибки в проекте, скорее всего, являются моими.

Что дальше? Мне кажется, что разработчики компилятора (см. Приложение B) должны попытаться уточнить и согласовать содержание этого документа, не добавлять в него ничего нового и, возможно, даже удалить из него некоторые пункты. Прежде всего, нам следует избегать повторения истории со стандартизацией Modula-2, поскольку на практике, вероятно, ничего полезного не получится.

Мы будем рады вашим отзывам о содержании проекта и возможности проведения следующей встречи.

Брайан Кирк

Робинсон
Ассошиэйтс Ред
Лайон Хаус
Сент-Мэри-стрит Пейнсвик
GLOS GL6 6QR
Голос (+ 44) (0)452 813 699
Факс (+ 44) (0)452 812 912
e-Mail: robinsons@cix.compulink.co.uk.

СОДЕРЖАНИЕ

[ЭТА СТРАНИЦА НАМЕРЕННО ОСТАВЛЕНА ПУСТОЙ]

Руководство Oakwood для разработчиков компилятора Oberon-2

1.0 Введение

1.1 Oakwood Guidelines

Эти рекомендации были подготовлены группой разработчиков компилятора Oberon-2, включая разработчиков ETH, после встречи в отеле Oakwood в Кройдоне, Великобритания, в июне 1993 года. Целью этой встречи было согласование стандартной спецификации языка Oberon- 2, некоторых минимальных расширений и стандартной переносимой библиотеки. Предполагается, что все разработчики должны обеспечить поддержку Oberon-2, по крайней мере, на уровне спецификации ETH, а также предложить реализацию основных библиотечных модулей. Цель состоит в том, чтобы гарантировать, что программы Oberon-2, использующие библиотеку, будут последовательными и переносимыми во всех формирующих реализациях.

Первоначальным мотивом встречи в Оуквуде было желание избежать повторения судьбы Modula-2 с коммерческими реализациями Oberon-2. К сожалению, в реализации Modula-2 появилось множество диалектов языка и множество несовместимых базовых библиотек. Процесс стандартизации Modula-2 занял слишком много времени и открыл дверь в ящик Пандоры для расширений. Целью данного отчета является признание отчета по языку ETH Oberon-2 в качестве базового стандарта и предоставление информации, полезной для разработчиков компиляторов, чтобы компиляторы и их базовые библиотеки обеспечивали базовый уровень совместимости.

1.2 Язык Oberon-2 Стандарт

Стандартная спецификация языка ETH Oberon-2 содержится в отчете, который контролируется и публикуется ETH Zrich. Текущая версия отчета доступна по анонимной FTP-пересылке через INTERNET из каталога :

neptune.inf.ethz.ch:~ ftp/Oberon/Docu

Последняя полная версия отчета ETH Oberon-2 находится в

файле Oberon2.Report.ps.Z.

Хронологический список всех изменений, внесенных в отчет,

находится в файле Oberon2.ChangeList.ps.z

1.3 Использование имени Oberon

Название Oberon было защищено торговой маркой ETH в контексте операционной системы и языка. В целях соблюдения торговой марки ETH любой компилятор, который, по крайней мере, не реализует ETH Oberon или Oberon-2, не должен упоминаться или называться компилятором Oberon или Oberon-2.

При упоминании возможностей ETH Oberon в документации допустимо использовать термины Oberon или Oberon-2. Однако при упоминании любых расширений, специфичных для компилятора, термин Oberon должен быть дополнен прилагательным.

Например: "XYZ Oberon-2 поддерживает комплексные числа".

В интересах пользователей настоятельно рекомендуется, чтобы каждый раз, когда реализатор предоставляет описание своего продукта, он указывал расширения, которые он поддерживает или не поддерживает, а также дополнительные библиотеки, которые он предоставляет.

Разработчики должны указывать в описании своих компиляторов, поддерживаются ли расширения в соответствии с данным Руководством Oakwood.

2.0 Язык Уточнения

2.1 Введение

Эта глава состоит из списка языковых пояснений. В идеале необходимость в пояснениях отпадает, и можно надеяться, что в будущем в отчет ETH Oberon-2 будут внесены соответствующие изменения. Приведенные здесь уточнения являются снимком ситуации, сложившейся в сентябре 93-го года.

2.2 Статус НИЛ

NIL - это зарезервированное слово, обозначающее предопределенное значение. В отличие от TRUE и FALSE, тип NIL не может быть выражен в Oberon-2.

2.3 Нелегальные операции

Следующие операции являются незаконными. Их эффект зависит от системы.

1. Отмена ссылки на указатель NIL.
2. Вызов переменных процедуры со значением NIL.
3. Тесты типов и защитники типов с указателями NIL.
4. Индексирование массива с индексом, выходящим за пределы диапазона.
5. Доступ к элементу набора за пределами диапазона 0 ... MAX (SET).
6. Применение SHORT (...) к аргументу, значение которого не входит в диапазон типа результата.
7. Операции над строками или символьными массивами, содержащими строки, которые не являются нулевыми.
8. Переполнения.

2.4 Переменные WITH и охраняемые

Можно изменить охраняемую переменную-указатель в пределах области действия охраняющего оператора WITH, например:

```
ТИП
 T = КОНЕЦ ЗАПИСИ; P = УКАЗАТЕЛЬ НА T;
 T1 = ЗАПИСЬ (T) END; P1 = УКАЗАТЕЛЬ НА
 T1; T2 = ЗАПИСЬ (T) END; P2 =
 УКАЗАТЕЛЬ НА T2;

ПРОЦЕДУРА X;
 VAR p: P; p1: P1; p2: P2;
```

```
PROCEDURE Y;
```
НАЧАТЬ

```
 p := p2
 END Y;
НАЧАТЬ
 NEW (p); NEW(p1); NEW(p2); p := p1;
 WITH p: P1 DO
 Y (*p теперь имеет тип P2, а не P1*)
 END
КОНЕЦ X;
```

Практический способ справиться с этим - :

Если компилятор уверен в безопасности, то предупреждение не выдается. Если есть сомнения, то выдайте предупреждение. Сложный компилятор может автоматически вставить дополнительные проверки защиты типа.

2.5 Сравнение строк

Строки всегда завершаются нулем. Массивы символов, которые будут сравниваться или использоваться в качестве исходного операнда процедуры COPY, должны содержать 0X в качестве терминатора.

Сравнение a relop b, где a и b - (открытые) символьные массивы или строки, a relop - это
=, #, >, >=, <, <= выполняется в соответствии со следующим псевдокодом

```
PROCEDURE Compare (a, b: ARRAY OF CHAR; relop:RELATION):
BOOLEAN;
 i := 0;
 WHILE (a[i] 1 0X) & (a[i]=b[i])
 DO
 INC (i)
 END;
 RETURN a[i] relop b[i]
END Compare
```

2.6 Рекурсивные объявления и импорт

Декларации

Объявление структурированного типа не может содержать само себя. Например, объявление RECORD не может иметь себя в качестве типа одного из своих полей.

Модуль не должен импортировать сам себя, например

```
МОДУЛЬ x;
 ИМПОРТ
```

```
 x;
END x.
```

Однако имя модуля можно использовать для псевдонимов, например

```
МОДУЛЬ x;
 ИМПОРТ
 x:=y;
 VAR i: x.INTEGER;
END x.
```

Однако это плохой стиль программирования.

2.7 Совместимость строк и символов

Строка длины 1 может быть использована в любом контексте, где разрешена символьная константа, и наоборот.

2.8 Повторное объявление заранее объявленных идентификаторов

Любой заранее объявленный идентификатор может быть объявлен заново. Например

```
ТИП INTEGER = LONGINT;
```

и

```
ПРОЦЕДУРА
АБС;
НАЧАТЬ
  ...
END ABS;
```

Очевидно, что от такой практики следует воздерживаться, а если она вообще используется, то с особой осторожностью.

2.9 Усечение точности

Иерархия включения типов может предполагать неявное усечение точности между REAL и LONGINT. Например, если оба типа представлены в 32 битах, то точность REAL, скорее всего, будет составлять только 24 бита. Поэтому присваивание из LONGINT в REAL будет включать усечение точности присваиваемого значения.

3.0 Язык Расширения

3.1 Введение

Расширения языка - это возможности, предоставляемые разработчиками компиляторов в дополнение к языку, указанному в отчете ETH Oberon-2.

Цель этой главы не в том, чтобы поощрять расширения. Причина их определения в том, чтобы способствовать единообразному подходу к спецификации и предоставлению расширений в разных компиляторах и постараться сделать так, чтобы одно и то же расширение поддерживалось несколькими компиляторами и имело одинаковый синтаксис и семантику в каждом из них. Если конкретный компилятор предлагает средства для опциональной поддержки расширений языка, то по умолчанию компиляция не должна включать никаких расширений.

3.2 Дополнительные типы данных

Расширение ETH Oberon-2 новыми типами данных - очень спорный вопрос. На встрече в Оук-Вуд общее мнение было таково, что только тип комплексного числа должен быть расширен. Перечисления и беззнаковые типы были специально отвергнуты ETH, хотя они по-прежнему считаются желательными для прикладных программистов. Беззнаковые типы особенно важны при взаимодействии с существующими внешними стандартными библиотеками, такими как X Windows, 'C' или Windows, и получили поддержку со стороны прикладных программистов. Типы битового уровня считаются ненужными, поскольку можно использовать тип SET.

3.2.1 Иерархии включения типов

Добавление типов данных, которые являются дополнительными для Oberon-2, должно осуществляться с пониманием (если оно вообще возможно) и с учетом последствий для всего языка. Для разделения семейств типов, которые по своей сути несовместимы, следует использовать отдельные иерархии включения типов. Для преобразования значений, которые могут быть представлены в разных иерархиях включения типов, следует использовать явные процедуры преобразования. Предопределенные функциональные процедуры LONG и SHORT должны обеспечивать преобразование в рамках любой расширенной иерархии типов.

Пример (пожалуйста, обратите внимание, что это НЕ предложение для общего применения)

```
LONGCOMPLEX ⇒ REAL ⇒ LONGINT ⇒ INTEGER ⇒ SHORTINT

LONGCARD ⇒ CARDINAL ⇒ SHORTCARD

LONGCHAR ⇒ CHAR
```

Цель этой схемы - сохранить преимущества включения типов, но при этом явно

разделить зависящие от системы аспекты преобразования значений между типами в различных иерархиях включения типов. Такие процедуры должны быть включены как встроенные процедуры. Для любого дополнительного расширения языка типами данных реализатор обязан предоставить обновленную версию отчета о языке Oberon-2 с указанием всех необходимых изменений.

Существует известная проблема с этим предложением. Оно не позволяет включить тип COM- PLEX в LONGREAL. Однако было сочтено, что это лучшее решение, чем иметь только один сложный тип, выбираемый как LONG переключателем компилятора, который может быть легко установлен для выбора различных вариантов в различных модулях (и библиотечных модулях).

<< BK: Необходимо предложение для процедуры преобразования, см. раздел 3.3.1 >>.

3.3 Тип COMPLEX и LONGCOMPLEX

Комплексные числа состоят из двух частей (действительная, мнимая). Тип LONGCOM- PLEX определяется как (LONGREAL, LONGREAL) и может быть включен в верхний конец иерархии включения типов. Тип COMPLEX определяется как (REAL, REAL) и является расширением REAL в иерархии. См. раздел 3.4.1. Если значение типа, меньшего или равного REAL, интерпретируется как значение COMPLEX, то оно считается действительной частью; соответствующая мнимая часть равна 0. К комплексным типам могут применяться все правила совместимости выражений и присваиваний, например

```
VAR
 c:
 КОМПЛЕКС;
 r: REAL;
 i: INTEGER;
 c:=i+r;
 c:=c*r;
```

3.3.1 Новые функции преобразования

Определены следующие заранее объявленные процедуры функций, (z) обозначает выражение

Имя	**Тип аргумента**	**Тип результата**	**Функция**
RE(z)	КОМПЛЕКТ	РЕАЛЬНЫЕ	Реальная часть
RE(z)	LONGCOMPLEX	LONGREAL	Реальная часть
IM(z)	КОМПЛЕКТ	РЕАЛЬНЫЕ	Воображаема я часть
IM(z)	LONGCOMPLEX	LONGREAL	Воображаема я часть

<< BK/HM/AF: Предъявленные функции SHORT, LONG, MIN, MAX и SIZE должны быть определены для COMPLEX и LONGCOMPLEX.

3.3.2 Синтаксис комплексного литерального числа

Для литералов комплексных чисел используется общая

нотация: number = integer | real | complex.

complex = real "i".

Примеры

```
 Значени
 я RE
 IM
1.i 0. 1.
2.+3.i 2. 3.
4. 4. 0.
5.3-6.2i 5.3-6.2
```

3.3.3 Причины против внедрения COMPLEX

Отсутствие типов данных COMPLEX в Oberon было преднамеренным решением ETH, а не недосмотром. Йозеф Темпл приводит следующие причины.

3.3.3.1 Внутреннее представительство

Картезианская или полярная? Оба варианта имеют свои преимущества, однако декартовый более распространен.

3.3.3.2 Эффективность

Наибольшей эффективности можно добиться, только кодируя КОМПЛЕКСНЫЕ операции как РЕАЛЬНЫЕ, потому что часто реальные или мнимые части равны нулю, единице или значению, которое позволяет алгебраически упростить их.

3.3.3.3 Точность

Не полностью контролируется программистом в случае COMPLEX.

3.3.3.4 Трудности в иерархии числовых типов.

Включение линейного типа изменится на направленный ациклический граф (DAG), если ввести два типа COMPLEX и LONGCOMPLEX с правилами совместимости, как это естественно ожидать.

Упрощением было бы задать COMPLEX = LONGCOMPLEX, но достаточно ли этого? Другим упрощением было бы формирование отдельной иерархии, состоящей из COMPLEX и LONGCOMPLEX, но удобно ли это?

Также следует обратить внимание на то, как это отразится на остальных определениях языка. Например, как быть с операторами сравнения для числовых типов?

3.3.3.5 Реализация

Не самый важный момент, но COMPLEX также усложняет компилятор, особенно когда нужно сгенерировать хороший код. Причина в том, что для представления

двух частей одного комплекса в компиляторе необходимо вести два отдельных дескриптора операндов.

3.3.3.6 Оборудование

В отличие от INTEGER и REAL, аппаратная поддержка COMPLEX отсутствует.

3.3.3.7 Синтаксис

Для обозначения сложных констант необходим дополнительный синтаксис и/или дополнительные предикатные функции.

3.3.3.8 Возврат структурированных функций

Распространенным заблуждением является то, что введение структурированных возвратов функций устранит дискуссию о COMPLEX, поскольку тогда можно будет определять сложные операции как функции. Следует отметить, что это только половина пути, поскольку математики все еще хотят иметь инфиксную нотацию, что потребует введения более общей концепции перегрузки, включающей инфиксные операторы. Это, в свою очередь, нарушило бы идею всегда квалифицировать импортируемые объекты по имени модуля.

3.3.3.9 Неиспользованный

По причинам, описанным выше (эффективность, точность), многие программисты на Фортране не используют сложные операции, хотя они и поддерживаются языком.

3.3.3.10 Недостаточно

Для научных вычислений COMPLEX - это лишь небольшой шаг. Не хватает еще векторных операций и субмассивов.

3.4 Процедуры прерывания и кода

Последовательное средство обеспечения чистого интерфейса Oberon-2 для сильно зависящих от системы функций определяется путем инкапсуляции таких функций в процедуры, которые по своей сути являются небезопасными.

Прерывания реализуются путем маркировки процедуры с помощью префикса +.

```
PROCEDURE +Proc ... ; ... END Proc;
```

Во время выполнения процедура должна быть связана с требуемым прерыванием с помощью механизма установки, такого как

Установите `(Proc,` номер`);`

Где число представляет собой позицию в векторной таблице или фактическое расположение адреса вектора, очевидно, что это зависит от реализации.

Кодовые процедуры реализуются путем обозначения процедуры знаком - в качестве префикса.

```
PROCEDURE -ProcHeading byte {", "byte};
```

Например

```
ПРОЦЕДУРА -Sigblock* (маска: SET): SET;
82H, 0, 20H, 109, 91H, 0D0H, 20H, 0;
```

<< ВК: Далее следует более читабельная альтернатива, предложенная Стивом Метцелером, лично мне она гораздо больше нравится. Необходимо принять решение. Это добавит два новых ключевых слова.

Прерывания реализуются путем маркировки процедуры ключевым словом INTERRUPT в качестве префикса

```
INTERRUPT PROCEDURE Proc ...; ... END Proc;
```

Аналогично кодовые процедуры реализуются путем маркировки процедуры ключевым словом CODE и предоставления ей тела, содержащего шестнадцатеричные коды байтов или инструкции уровня ассемблера.

```
CODE PROCEDURE Proc;
BEGIN
 Байт {"," байт} | {Инструкции ассемблера}
END Proc;
```

3.5 Взаимодействие с внешними библиотеками

Когда программы Oberon-2 пишутся для внешних операционных систем, отличных от Oberon System, требуется механизм, обеспечивающий максимально бесшовный интерфейс между ними. Во избежание снижения производительности крайне желательно прямое сопоставление структур и соглашений Oberon-2 с внешними структурами и соглашениями. Также желательно, чтобы используемая нотация была практичной как для больших библиотек, так и для отдельных процедур внутри модуля. Признано, что использование внешнего механизма сопряжения делает модуль небезопасным.

Рекомендуется, чтобы в интересах студентов и начинающих изучать язык, в документацию механизма было включено предупреждение о вреде здоровью в стандартной форме, например, (** НЕ БЕЗОПАСНО**)

Четыре элемента, которые должны быть предусмотрены для сопряжения с внешними библиотеками, это

- название Oberon-2 для объекта
- тип "Оберон-2" и подпись объекта
- внешнее название объекта
- имя местоположения и стиль соглашения о вызове библиотеки или объекта.

Следующее предложение не было полностью опробовано, однако оно предлагается в

качестве основы для обсуждения.

Для модулей, содержащих множество процедур, принадлежащих одной библиотеке, синтаксис может быть следующим

```
MODULE OberonModuleName "[" convention "]" EXTERNAL "["
externalLibraryName "]" ...
```

Где условным обозначением может быть "PASCAL" или "C", a externalLibraryName также является строкой, заключенной в кавычки.

Например:

```
MODULE ISOStrings [ "Modula-2" ] EXTERNAL
[ "server_XP/lib" ] ...
```

За обычными идентификаторами Oberon-2 в модуле по желанию следует эквивалентное внешнее имя в виде строки, например:

```
PROCEDURE CreateWindow ["CREATE_WINDOW$BIG"] (... );
```

Обратите внимание, что внешние идентификаторы или строки, не относящиеся к Oberon-2, могут содержать любые символы, допустимые для внешней библиотеки.

Для модуля Oberon-2, который содержит всего одну или несколько интерфейсных процедур или скрывает структуру набора внешних модулей, можно использовать следующую форму.

```
PROCEDURE "["<конвенция>", "<имя внешней библиотеки>"]" ...
```

Например:

```
PROCEDURE ["C", "Motif.lib"] CreateWindow
 ["CREATE_WINDOW"] (... );
```

<< ВК: Пожалуйста, обратите внимание, что Йозеф Темпл и профессор Мёссенбёк из ETH категорически против того, чтобы разделы 3.6 и 3.7 предлагались в качестве расширений языка.

3.6 Подчеркивания в идентификаторах

Идентификаторы могут содержать

дополнительный символ "_" ident = (letter | "_")

{буква | цифра | "_"}.

Этот синтаксис допускает, чтобы идентификаторы начинались с символа подчеркивания "_".

3.7 Встроенный Экспоненциация

Оператор экспоненцирования "**" обеспечивает удобную нотацию для

арифметических выражений вместо использования вызовов функций. Это арифметический оператор, который имеет более высокий приоритет, чем операторы умножения и деления. В выражении

```
a := b**c ;
```

значение результата - это значение b, возведенное в степень при значении c.

<< BK: Это вводит в язык пятый уровень старшинства. Если мы включим его вообще, то необходимо определить полную грамматику выражений, чтобы она могла быть реализована последовательно. Пожалуйста, есть добровольцы...

4.0 Компиляция Контроль

4.1 Введение

Существует два основных вопроса, касающихся управления процессом компиляции: установка опций компиляции для компилятора и выбор конкретного исходного текста для компиляции. Существуют также две основные школы взглядов на то, как этот контроль должен быть задан. Прикладные программисты и руководители проектов часто предпочитают иметь один исходный текст, особенно когда программа разрабатывается с учетом множества вариантов (например, компилятор с очень похожими генераторами кода для семейства процессоров). Другие предпочитают использовать препроцессоры, чтобы сначала извлечь исходный текст определенного варианта, а затем скомпилировать его. На рынке наблюдается тенденция интеграции препроцессоров в компиляторы, основными причинами которой являются

- читаемость программы для сопровождающих, возможность видеть связи между переменными.
- прямая корреляция между сообщениями об ошибках компилятора и исходным текстом
- чтение исходного текста только один раз в течение всего процесса компиляции (для повышения скорости)
- экономия при хранении (без промежуточных версий)

Это, по-видимому, очень эмоциональный вопрос, поскольку различные организации твердо придерживаются своего конкретного подхода. В этой главе определены некоторые условные обозначения для контроля компиляции с намерением, чтобы производители компиляторов предлагали такие возможности на основе одной и той же базовой модели. Следует признать, что выбор нотации может быть предписан используемой операционной системой или соответствовать соглашениям, используемым в существующих компиляторах. Но даже в этом случае, если есть возможность следовать рекомендациям и уменьшить разнообразие, ее следует использовать.

Дополнительные языковые конструкции, определенные ниже, не следует рассматривать как часть языка Oberon-2. Скорее, они определяют отдельный язык управления компилятором, который сосуществует с языком Oberon-2 и отличается от него.

Все строчные команды компилятору содержатся в псевдокомментариях в стиле ISO с использованием угловых скобок <* ... *>.

4.2 Проверки во время выполнения

Проверки во время выполнения контролируются прагмами, которые используются для выборочного включения и выключения каждой опции. Все прагмы должны использоваться по умолчанию для обеспечения максимальной безопасности.

Синтаксис: "<*$"

{модификатор} "*>", где

модификатор - это

- pragma -set pragma OFF, disable
- pragma +set pragma ON, enable

- < складывать текущее состояние прагмы
- > снять текущее состояние pragma
- ! вернуться к состоянию pragma, заданному в исходной командной строке.

Следующие буквы взяты из компилятора ETH OP2 и приведены только в качестве ориентира. На практике они, скорее всего, будут специфичны для конкретной реализации по другим причинам совместимости (например, другие компиляторы, Unix ...)

pragma	**по умолчанию**	**значение**
A	+	Генерация ASSERT
K	+	Проверка переполнения стека
P	+	Инициализация указателя
R	+	Проверка диапазона (например, SHORT (Int) находится в диапазоне SHORTINT)
S	-	Разрешите файлу символа заменять предыдущую версию, если она отличается от предыдущей
T	+	Проверка типа (подавление защит типа)
V	+	Проверка перелива
X	+	Проверка индексов, как статических, так и динамических

Исходные прагмы могут быть как в верхнем, так и в нижнем регистре.

Примечание: В компиляторах ETH по умолчанию используется значение - для прагм R и V.

4.3 Опция компилятора управление

Опции компилятора можно включать и выключать с помощью операторов. Поскольку они применяются ко всей единице компиляции, имеет смысл использовать их только в начале модуля.

<*OPTION+ *>, чтобы включить OPTION и включить его
<*OPTION- *> для установки OPTION в

положение off, отключая его Например

<*STANDARD+ *>

Возможны следующие варианты

Вариант	**по умолча**	**значение**

	нию	
СТАНДАРТ	+	Стандарт отчета Oberon-2, без расширений
ИНИЦИАЛИЗИРОВАТЬ	+	Все указатели инициализируются
ГЛАВНАЯ	+	Генерирует точку входа в программу. Только одна на систему!
ПРЕДУПРЕЖДЕНИЯ	+	Сообщайте о сомнительном использовании

Примечание: Не существует возможности управления сборкой мусора, например, для систем, которым требуется детерминированное время. Этого можно добиться, явно вызвав системный менеджер памяти для выключения и включения сборки мусора. Также см. раздел 5.6.

4.4 Исходный текст компилятора контроль

Для больших программ, где один исходный текст должен поддерживать множество вариантов исполнения, возникает практическая необходимость в выборочной компиляции исходного текста. Выбор может осуществляться либо с помощью препроцессора, либо, в целях оптимизации дискового пространства, скорости и эффективности, на этапе компиляции.

Синтаксис для выражения выбора исходного текста следующий

```
<* IF condition THEN *>
<* ELSIF условие THEN *>
<* ELSE *>
<* END *>
```

Условное выражение состоит из определенных программистом ВЫБОРОВ, которые могут быть объединены в Oberon-подобное булево выражение, которое может содержать операторы ~, &, OR. Опции компилятора по сути являются предопределенными селекторами и могут быть использованы в условной части

Чтобы определить новый SELECTOR, который по умолчанию имеет значение FALSE

```
<* NEW SelectorName *>
```

Чтобы придать значение элементу SELECTOR

```
<* SelectorName+ *>, чтобы установить значение TRUE
<* SelectorName- *>, чтобы установить значение FALSE
```

Примеры:

```
<*IF ~ MAIN THEN *> ...

<*ЕСЛИ M68000 & WARNINGS THEN *>
 IMPORT CG68000;
<*ELSE *>
 IMPORT CG80x86;
<*END*>
```

5.0 Реализация Рекомендации

5.1 Введение

В этой главе приведены рекомендации, описывающие некоторые специфические характеристики для ком- пиляторов, которые соответствуют данным рекомендациям.

5.2 Тип диапазоны

Минимальное значение, возвращаемое MAX (тип), и максимальное значение, возвращаемое MIN (тип), должны быть как минимум (как максимум) следующими

ТИП	**'MAX' СТОИМОСТЬ**	**'MIN' СТОИМОСТЬ**
SHORTINT	127	-128
INTEGER	32767	-32768
LONGINT	+2147483647	-2147483648

По возможности, формат REALIEEE 32 бит
LONGREAL не менее точности формата REAL IEEE, по возможности более высокое разрешение SET32 элементов минимум (0..31)
CHAR0 ...0FFX, где ...
00..7FX ASCII-код
80 ..0FFX ISO LATIN-1 CODE предпочтительно, но набор кодов не определен

5.3 Расширение типа Уровни

Если реализация накладывает ограничение на количество уровней расширения типа, то оно не должно быть меньше 8 уровней, включая базовый тип.

5.4 Модуль SYSTEM

Система модулей должна быть основана на модели ETH, где это целесообразно.

5.5 Процедура SYSTEM.MOVE

Для процедуры SYSTEM.MOVE следует уточнить поведение при перекрытии исходного и конечного экстентов в отношении перезаписи, а также особый случай, когда длина = 0.

5.6 Сбор мусора

Автоматическая сборка мусора рекомендуется везде, где это возможно. Если

сборка мусора недоступна или имеется механизм для ее активации и деактивации, то процедура

DISPOSE может быть предоставлен в системе модулей. Он принимает один параметр, который является параметром значения указателя.

5.7 Реализация характеристики

Каждая реализация компилятора неизбежно имеет ограничения, например, по длине идентификатора или предоставляемым проверкам во время выполнения. Для каждой реализации может быть предоставлен список характеристик, чтобы пользователи могли судить о ее пригодности и проблемах переносимости, которые могут возникнуть при переходе от одной реализации к другой.

Определены следующие характеристики

Длина идентификатора, возможно не менее 23 значащих

символов Уровни расширения записи, 8, включая базовый тип

Фактические размеры типов (INTEGER, LONGINT, ...), см. 5.2

5.8 Инициализация указателей

Все указатели на процедурные переменные, переменные, поля записей и элементы массивов должны быть инициализированы компилятором безопасным значением, рекомендуется значение NIL. Это относится к указателям, которые выделяются статически, динамически или в стеках. Обратитесь к списку изменений ETH. (Раздел 1.2).

В отчете ETH не определено, что переменные инициализируются, однако практическая реализация может включать следующий подход ...

Компилятор должен гарантировать, что переменные уровня 0 любого типа указателя или процедуры статически или динамически инициализируются в NIL до выполнения инициализационной части модуля (тела модуля).

Компилятор должен предоставить код для динамической инициализации локальных переменных любого указателя или типа процедуры в NIL перед выполнением тела процедуры.

При выполнении заранее объявленной процедуры NEW менеджер хранилища/кучи системы времени выполнения (если таковой имеется) должен инициализировать пространство кучи в NIL. В качестве альтернативы компилятор должен выдать код для инициализации динамических переменных любого указателя или типа процедуры в NIL.

Может быть предусмотрен переключатель компиляции, запрещающий генерацию кода инициализации для переменных типа указателя или процедуры. В случае динамических переменных, выделенных с помощью процедуры NEW, может быть предусмотрен альтернативный модуль хранения (или системы времени

выполнения), который не выполняет инициализацию.

5.9 Обработка неопределенной семантики

Когда происходят операции с неопределенной семантикой, перечисленные в разделе 2.3, то их эффект зависит от системы и должен обрабатываться согласованно в рамках конкретной реализации. Ожидается, что программа завершится сообщением с указанием причины и ее программного местоположения.

5.10 Монадический '- '

В документации, поставляемой с компиляторами, должно быть ясно, что монадическое отрицание является оператором сложения и имеет более низкий приоритет, чем оператор умножения. Например, выражение -5 MOD 3 эквивалентно -(5 MOD 3).

5.11 Преобразование из целого числа в вещественное

Пользователям компилятора должно быть ясно, что функция LONG не может быть использована для приведения выражения типа LONGINT к типу REAL. Для этого не существует явной функции. Присваивание вида

real := integer;

необходимо использовать, который автоматически преобразует любой целочисленный тип в тип REAL.

5.12 Экспортировано Комментарии

Экспортированный комментарий обозначается двумя звездочками подряд после открывающей скобки, например

(** это экспортированный комментарий *)

Он сигнализирует браузеру, что комментарий должен быть включен в модуль DEFINITION, являющийся производным от обрабатываемого модуля. Это скорее соглашение, чем языковая проблема.

5.13 Только чтение VAR Параметры

Было много просьб сделать параметры ARRAY и RECORD доступными только для чтения, чтобы добиться эффективности передачи по ссылке без связанной с этим возможности повреждения вызывающего параметра. Попытка выполнить присваивание любому компоненту такого параметра, доступного только для чтения, является ошибкой времени компиляции. Такие параметры могут быть помечены стандартным символом "-", предназначенным только для чтения. Например:

```
PROCEDURE Print (theText-: ARRAY OF CHAR) ;
```

Обсуждения с ETH показывают, что это действительно проблема оптимизации кода компилятора, и на этом основании рекомендуется не реализовывать это расширение.

5.14 Тип Охранники с параметрами RECORD

Если запись присваивается формальной записи параметра VAR, компилятор должен сгенерировать неявный тест типа, чтобы убедиться, что статический тип и динамический тип записи назначения совпадают.

6.0 Библиотека Модули

6.1 Введение

Для программистов очень важно, чтобы базовый набор библиотечных модулей был доступен в различных реализациях компилятора. С другой стороны, очевидно, что невозможно разработать библиотечные модули, которые были бы полезны для всех целей. Чтобы быть эффективными, библиотечные модули должны иметь цель, которая имеет смысл для пользователя библиотеки.

В этом отчете определены две группы модулей

- модули, основанные на проектах ETH Oberon System, которые обеспечивают возможности ввода-вывода и поддержку опубликованных учебных материалов, в частности, серии книг Oberon от авторов ETH
- модули, расширяющие функциональность языка стандартным образом, например, математические библиотеки.

Определения модулей, приведенные в Приложении 1, призваны стимулировать всех разработчиков компиляторов предлагать наборы библиотечных модулей с одинаковым интерфейсом и функциональностью.

Все реализации должны поддерживать все так называемые базовые модули, описанные в разделе
6.2. Дополнительные модули должны быть предоставлены, если они имеют отношение к конкретной реализации ком- пилятора (например, если поддерживается COMPLEX).

6.2 Основные модули

Предполагается, что базовые модули будут поставляться со всеми реализациями компилятора Oberon-2. Они основаны на разработках ETH Oberon System ...

- XYplaneЭлементарное пиксельное черчение
- ВводДоступ к клавиатуре и указателю
- Ввод из стандартного потока
- ВыходВывод в стандартный поток
- ФайлыФайл ввода вывода, с всадниками
- СтрокиПростые манипуляции со строками
- МатематикаМатематические и триггерные функции для REAL
- MathLMath и триггерные функции для LONGREAL

6.3 Дополнительные модули

Дополнительные модули предоставляются компилятору по м е р е необходимости...

- CoroutinesПредставляет собой невытесняющие потоки, каждый из которых имеет собственный стек, но все совместно используют общее адресное пространство.
- Функции MathCMaths для COMPLEX
- MathLCМатематические функции для LONGCOMPLEX

Приложение A: Библиотечные модули

1.1 Базовая библиотека Модули

Ожидается, что все реализации компилятора Oberon-2 будут включать следующие модули ...

XYplane Вход Вход Выход Файлы СтрокиMath и MathL

1.2 Дополнительные модули

Следующие модули являются необязательными. Если они предусмотрены, то должны соответствовать спецификациям ...

- Корутины
- MathC и MathLC

1.2.1 Модуль XYplane

Модуль XYplane предоставляет некоторые базовые возможности для программирования графики. Его интерфейс максимально упрощен, поэтому он больше подходит для упражнений по программированию, чем для серьезных графических приложений.

XYplane - это декартова плоскость пикселей, которые можно рисовать и стирать. Плоскость отображается на некоторое место на экране. Переменные X и Y указывают ее левый нижний угол, W - ширину, H - высоту. Все переменные доступны только для чтения.

```
ОПРЕДЕЛЕНИЕ XYplane;
CONST draw = 1; erase = 0;
VAR X, Y, W, H: INTEGER;
PROCEDURE Open;
ПРОЦЕДУРА Очистить;
PROCEDURE Dot (x, y, mode: INTEGER);
PROCEDURE IsDot (x, y: INTEGER): BOOLEAN;
PROCEDURE Key (): CHAR;
END XYplane.
```

1.2.1.1 Операции

Open инициализирует плоскость рисунка.

Clear стирает все пиксели в плоскости рисунка.

Dot(x, y, m) рисует или стирает пиксель в координатах (x, y) относительно левого

нижнего угла плоскости. Если m=draw, то пиксель рисуется, если m=erase, то стирается.

IsDot(x, y) возвращает TRUE, если нарисован пиксель в координатах (x, y) относительно левого нижнего угла экрана, в противном случае возвращается FALSE.

Key() считывает данные с клавиатуры. Если клавиша была нажата до вызова, возвращается ее символьное значение, в противном случае результат равен 0X.

1.2.1.2 Примечания

В системе ETH Oberon System Open открывается вьювер, который занимает весь путь пользователя. В качестве контуров этого вьювера используется плоскость рисования, предоставляемая XYplane.

1.2.1.3 Происхождение

Разработан Мартином Райзером для книги "Программирование в Oberon". Приведенная выше спецификация была предложена Х. Мёссенбёком, ETH

1.2.2 Вход модуля

Модуль ввода обеспечивает доступ к мыши, клавиатуре и часам.

```
ОПРЕДЕЛЕНИЕ Вход;
 VAR TimeUnit: LONGINT;
PROCEDURE Available (): INTEGER;
PROCEDURE Read (VAR ch: CHAR);
PROCEDURE Mouse (VAR keys: SET; VAR x, y: INTEGER);
PROCEDURE SetMouseLimits (w, h: INTEGER);
PROCEDURE Time (): LONGINT;
 Вход END.
```

1.2.2.1 Государство

Буфер клавиатуры. Очередь символов, вводимых с клавиатуры.

Время. Прошедшее время с момента запуска системы в единицах размера 1/TimeUnit секунд.

1.2.2.2 Операции

Available() возвращает количество символов в буфере клавиатуры.

Read(ch) возвращает (и удаляет) следующий символ из буфера клавиатуры. Если буфер пуст, Read ждет, пока не будет нажата клавиша.

Mouse(keys, x, y) возвращает текущую позицию мыши (x, y) в пикселях относительно левого нижнего угла экрана. keys - это набор нажатых в данный

момент клавиш мыши (left = 2, mid- dle = 1, right = 0).

SetMouseLimits(w, h) задает прямоугольник, в котором перемещается мышь (в пикселях). Последующие вызовы операции Mouse будут возвращать координаты x в диапазоне 0..w-1 и y в диапазоне 0..h-1.

Time() возвращает время, прошедшее с момента запуска системы, в единицах размера 1/TimeUnit секунд.

1.2.2.3 Примеры

```
IF Input.Available() > 0 THEN Input.Read(ch) END;
 REPEAT
 Input.Mouse(keys, x, y);
 ... нарисовать курсор мыши в позиции (x, y) ...
UNTIL keys = {}
секунды := Вход.Время() DIV Вход.Единица времени
```

1.2.2.4 Происхождение

Часть системы ETH Oberon. Приведенная выше спецификация была предложена X. Мёссенбёком, ETH.

1.2.3 Модуль B

Модуль In предоставляет набор базовых процедур для форматированного ввода символов, последовательностей символов, чисел и имен. Он предполагает наличие стандартного потока ввода с текущей позицией, которая может быть сброшена в начало потока.

```
ОПРЕДЕЛЕНИЕ B;
  VAR Done: BOOLEAN;
 PROCEDURE Open;
  PROCEDURE Char (VAR ch: CHAR);
  PROCEDURE Int (VAR i: INTEGER);
  PROCEDURE LongInt (VAR i: LONGINT);
  PROCEDURE Real (VAR x: REAL);
  PROCEDURE LongReal (VAR y: LONGREAL);
  PROCEDURE String (VAR str: ARRAY OF CHAR);
  PROCEDURE Name (VAR name: ARRAY OF CHAR);
 END In.
```

1.2.3.1 Государство

Текущая позиция. Позиция символа во входном потоке, из которого считывается следующий символ. При открытии (пере)устанавливается в начало входного потока. После чтения символа текущая позиция устанавливается в позицию, следующую сразу за этим символом. До первого вызова Open текущая позиция не

определена.

Готово. Указывает на успех операции ввода. Если после выполнения операции ввода значение Done равно TRUE, то операция была успешной и ее результат действителен. Неудачная операция ввода устанавливает значение Done в FALSE; оно остается FALSE до следующего вызова Open. В частности, Done устанавливается в FALSE, если предпринимается попытка чтения за пределы конца входного потока.

1.2.3.2 Операции

Open (повторно) устанавливает текущую позицию в начало входного потока. Done указывает, была ли операция успешной.

Следующие операции требуют Done = TRUE и гарантируют (Done = TRUE и результат действителен) или (Done = FALSE). Все операции, кроме Char, пропускают ведущие пробелы, табуляции или символы конца строки.

Char(ch) возвращает символ ch в текущей позиции.

LongInt(n) и Int(n) возвращают (длинную) целую константу n в текущей позиции в соответствии с форматом:

IntConst = digit {digit} | digit {hexDigit} "H".

Real(n) возвращает вещественную константу n в текущей позиции в

соответствии с форматом: RealConst = digit {digit} [{digit} ["E" ("+" | "-")

цифра {цифра}]].

LongReal(n) возвращает длинную вещественную константу n в текущей позиции в соответствии с фор-матом:

LongRealConst = digit {digit} [{digit} [("D" | "E")

("+" | "-") digit {digit}]].

String(s) возвращает строку s в текущей позиции в соответствии с

форматом: StringConst = "' char {char} "'.

Строка не должна содержать символов, меньших, чем пробел, таких как EOL или TAB.

Name(s) возвращает имя s в текущей позиции в соответствии с форматом имени файла базовой операционной системы (например, "lib/My.Mod" под Unix).

Пример:

```
VAR i: INTEGER; ch: CHAR; r: REAL; s, n: МАССИВ 32
СИМВОЛОВ;
```

```
...
In.Open;
In.Int(i); In.Char(ch); In.Real(r); InString(s);In.Name(n)
```

Входной поток:

```
  123*1.5 "abc" Mod.Proc
```

Результаты:

```
  i = 123
 ch = "*"
 r = 1,5E0
 s = "abc"
 n = "Mod.Proc"
```

1.2.3.3 Примечания

В системе ETH Oberon входной поток - это текст, следующий сразу за последней вызванной командой. Если этот текст начинается с символа "^", то текущая позиция устанавливается на начало последнего выделения (если выделения нет, то Done = FALSE). Если текст начинается с символа "*", то текущая позиция устанавливается на начало текста в отмеченном вьювере (если вьювер не отмечен, Done = FALSE). Конец входного потока - это конец текста, содержащего текущую позицию. Ввод коротких целых чисел не предусмотрен.

1.2.3.4 Происхождение

Разработан Мартином Райзером для книги "Программирование в Oberon". Приведенная выше спецификация была предложена Х. Мёссенбёком, ETH.

1.2.4 Модуль Out

Модуль Out предоставляет набор базовых процедур для форматированного вывода символов, чисел и строк. Предполагается наличие стандартного потока вывода, в который записываются символы.

```
ОПРЕДЕЛЕНИЕ  Выйти;
 ПРОЦЕДУРА  Открыть;
 PROCEDURE Char (ch: CHAR);
 PROCEDURE String (str: ARRAY OF CHAR);
 PROCEDURE Int (i, n: LONGINT);
 PROCEDURE Real (x: REAL; n: INTEGER);
 PROCEDURE LongReal (x: LONGREAL; n: INTEGER);
 PROCEDURE Ln;
END Out.
```

1.2.4.1 Операции

Open инициализирует выходной поток.

Char(ch) записывает символ ch в конец выходного потока.

String(s) записывает в конец выходного потока последовательность символов s с нулевым окончанием (без 0X).

Int(i, n) записывает целое число i в конец выходного потока. Если для текстового представления i требуется m символов, то i выравнивается вправо в поле из Max(n, m) символов с пробелами в левом конце. Знак плюс не записывается.

Real(x, n) записывает вещественное число x в конец выходного потока, используя экспоненциальную форму. Если текстовое представление x требует m символов (включая двузначную знаковую экспоненту), то x корректируется вправо в поле из Max(n, m) символов, заполненное пробелами с левого конца. Знак плюс мантиссы не записывается.

LongReal(x, n) записывает длинное вещественное число x в конец выходного потока, используя экспоненциальную форму. Если текстовое представление x требует m символов (включая трехзначную знаковую экспоненту), то x корректируется вправо в поле из Max(n, m) символов, заполненное пробелами с левого конца. Знак плюс мантиссы не записывается.

Ln записывает символ конца строки в конец выходного

потока. Примеры

	вывод (звездочками обозначены пустые места)
	`Out.Open;`
`Out.Int(-3, 5);`	`***3`
`Out.Int(3, 0);`	`3`
`Out.Real(1.5, 10);`	`**1.50E+00`
`Out.Real(-0.005, 0)`	`-5.0E-03`

1.2.4.2 Примечания

В системе ETH Oberon вывод добавляется в текст вывода, который очищается при загрузке модуля Out. Текст вывода может быть отображен в новом окне просмотра вызовом процедуры Open (Open также может быть вызван как команда).

1.2.4.3 Происхождение

Разработан Мартином Райзером для книги "Программирование в Oberon". Приведенная выше спецификация была предложена Х. Мёссенбёком, ETH.

1.2.5 Файлы модулей

Модуль Files обеспечивает операции над файлами и каталогом файлов.

```
DEFINITION Files;
  IMPORT SYSTEM;
```

```
ТИП
File = POINTER TO Handle;
Rider = RECORD
  eof: BOOLEAN;
  res: LONGINT;
КОНЕЦ;
PROCEDURE Old (name: ARRAY OF CHAR): File;
PROCEDURE New (name: ARRAY OF CHAR): Файл;
PROCEDURE Register (f: File);
PROCEDURE Close (f: File);
PROCEDURE Purge (f: File);
PROCEDURE Delete (name: ARRAY OF CHAR; VAR res: INTEGER);
PROCEDURE Rename (old, new: ARRAY OF CHAR;
                  VAR res: INTEGER);
PROCEDURE Length (f: File): LONGINT;
PROCEDURE GetDate (f: File; VAR t, d: LONGINT);
PROCEDURE Set (VAR r: Rider; f: File; pos: LONGINT);
PROCEDURE Pos (VAR r: Rider): LONGINT;
PROCEDURE Base (VAR r: Rider): Файл;
PROCEDURE Read (VAR r: Rider; VAR x: SYSTEM.BYTE);
PROCEDURE ReadInt (VAR R: Rider; VAR x: INTEGER);
PROCEDURE ReadLInt (VAR R: Rider; VAR x: LONGINT);
PROCEDURE ReadReal (VAR R: Rider; VAR x: REAL);
PROCEDURE ReadLReal (VAR R: Rider; VAR x: LONGREAL);
PROCEDURE ReadNum (VAR R: Rider; VAR x: LONGINT);
PROCEDURE ReadString (VAR R: Rider; VAR x: ARRAY OF CHAR);
PROCEDURE ReadSet (VAR R: Rider; VAR x: SET);
PROCEDURE ReadBool (VAR R: Rider; VAR x: BOOLEAN;
PROCEDURE ReadBytes (VAR r: Rider;
                     VAR x: ARRAY OF SYSTEM.BYTE;
                     n: LONGINT);
PROCEDURE Write (VAR r: Rider; x: SYSTEM.BYTE);
PROCEDURE WriteInt (VAR R: Rider; x: INTEGER);
PROCEDURE WriteLInt (VAR R: Rider; x: LONGINT);
PROCEDURE WriteReal (VAR R: Rider; x: REAL);
PROCEDURE WriteLReal (VAR R: Rider; x: LONGREAL);
PROCEDURE WriteNum (VAR R: Rider; x: LONGINT);
PROCEDURE WriteString (VAR R: Rider; x: ARRAY OF CHAR);
PROCEDURE WriteSet (VAR R: Rider; x: SET);
PROCEDURE WriteBool (VAR R: Rider; x: BOOLEAN);
```

```
PROCEDURE WriteBytes (VAR r: Rider;
                      VAR x: ARRAY OF SYSTEM.BYTE;
                      n: LONGINT)
END Files.
```

1.2.5.1 Типы

Файл представляет собой поток байтов, обычно хранящийся на внешнем носителе. Он имеет определенную длину, а также дату и время последней модификации.

Каталог файлов - это отображение имен файлов на файлы. Файл, не зарегистрированный в каталоге, считается временным.

Rider - это позиция чтения/записи в файле (позиции начинаются с 0). Для одного и того же файла может быть установлено несколько райдеров. Поле eof устанавливается в TRUE, если была предпринята попытка чтения за пределы конца файла. Поле res сообщает об успехе операций ReadBytes и WriteBytes. При записи данных старые данные перезаписываются в позиции райдера. При записи данных за пределы конца файла длина файла увеличивается.

1.2.5.2 Операции над файлами и каталогом файлов

Old(fn) ищет имя fn в каталоге и возвращает соответствующий файл. Если имя не найдено, возвращается NIL.

New(fn) создает и возвращает новый файл. Имя fn запоминается для последующего использования операции Register. Файл попадает в каталог только при вызове Register.

Register(f) заносит файл f в каталог вместе с именем, указанным в операции New, создавшей f. Файловые буферы записываются обратно. Любое существующее сопоставление этого имени с другим файлом перезаписывается.

Close(f) записывает обратно файловые буферы файла f. Файл по-прежнему доступен по его хэндлу f и расположенным на нем райдерам. Если файл не изменяется, закрывать его не нужно.

Purge(f) обнуляет длину файла f до 0.

Delete(fn, res) удаляет запись в каталоге для файла fn, не удаляя сам файл. Если res=0, то файл успешно удален. Если во время вызова Delete в файле были переменные, ссылающиеся на него, они все еще могут быть использованы.

Rename(oldfn, newfn, res) переименовывает запись каталога oldfn в newfn. Если res=0, файл успешно переименован. Если во время вызова Rename в файле были переменные, ссылающиеся на него, они все еще могут быть использованы.

Length(f) возвращает количество байт в файле f.

GetDate(f, t, d) возвращает время t и дату d последней модификации файла f. Кодировка следующая: час = t DIV 4096; минута = t DIV 64 MOD 64; секунда = t MOD 64; год = d DIV 512; месяц = d DIV 32 MOD 16; день = d MOD 32.

1.2.5.3 Операции на всадниках

Set(r, f, pos) устанавливает всадник r на позицию pos в файле f. Поле r.eof устанавливается в FALSE. Операция требует, чтобы 0 <= pos < Length(f).

Pos(r) возвращает позицию всадника r.

Base(r) возвращает файл, для которого установлен rider r.

1.2.5.4 Операции для неформатированного ввода и вывода

В общем случае все операции должны использовать следующий формат для внешнего представления:

Представление "Little endian" (т.е. младшим байтом слова является тот, который имеет наименьший адрес в файле).

Числа: SHORTINT 1 байт, INTEGER 2 байта, LONGINT 4 байта

Наборы: 4 байта, элемент 0 - наименее значимый бит

Булевые числа: один байт с FALSE = 0, TRUE = 1

Вещества: Стандарт IEEE; REAL - 4 байта, LONGREAL -

8 байт Строки: с окончанием 0X

1.2.5.5 Чтение

Read(r, x) считывает следующий байт x из райдера r и соответствующим образом продвигает r.

ReadInt(r, i) и ReadLInt(r, i) считывают целое (длинное) число i из райдера r и продвигают r соответственно.

ReadReal(r, x) и ReadLReal(r, x) считывают (длинное) вещественное число x из всадника r и продвигают r соответствующим образом.

ReadNum(r, i) считывает целое число i из райдера r и соответствующим образом продвигает r. Число i компактно закодировано (см. замечания ниже).

ReadString(r, s) считывает последовательность символов (включая завершающий 0X) из райдера r и возвращает ее в s. Райдер продвигается соответствующим образом. Фактический параметр, соответствующий s, должен быть достаточно длинным, чтобы вместить последовательность символов плюс завершающий 0X.

ReadSet(r, s) считывает набор s из всадника r и соответствующим образом продвигает r.

ReadBool(r, b) считывает булево значение b из всадника r и соответствующим образом продвигает r.

ReadBytes(r, buf, n) считывает n байт в буфер buf, начиная с позиции райдера r. Райдер продвигается вперед соответствующим образом. Если удалось прочитать меньше n байт, r.res содержит количество запрошенных, но непрочитанных байт.

1.2.5.6 Написание

Write(r, x) записывает байт x в райдер r и соответствующим образом продвигает r.

WriteInt(r, i) и WriteLInt(r, i) записывают (длинное) целое число i в райдер r и продвигают r соответственно.

WriteReal(r, x) и WriteLReal(r, x) записывают (длинное) вещественное число x в райдер r и продвигают r соответственно.

WriteString(r, s) записывает последовательность символов s (включая завершающий 0X) в райдер r и соответствующим образом продвигает r.

WriteNum(r, i) записывает целое число i в райдер r и соответствующим образом продвигает r. Число i компактно закодировано (см. замечания ниже).

WriteSet(r, s) записывает набор s в райдер r и соответствующим образом продвигает r.

WriteBool(r, b) записывает булево значение b в райдер r и соответствующим образом продвигает r.

WriteBytes(r, buf, n) записывает первые n байт из buf в райдер r и соответствующим образом продвигает r. r.res содержит количество байт, которые не удалось записать (например, из-за ошибки заполнения диска).

1.2.5.7 Примеры

```
VAR f: Files.File; r: Files.Rider; ch: CHAR;
```

Чтение из существующего файла xxx:

```
f := Files.Old("xxx");
IF f # NIL THEN
 Files.Set(r, f, 0);
 Files.Read(r, ch);
 WHILE ~ r.eof DO ... Files.Read(r, ch) END
END
```

Запись в новый файл yyy:

```
f := Files.New("yyy");
Files.Set(r, f, 0);
Files.WriteInt(r, 8); Files.WriteString(r, " bytes");
Files.Register(f)
```

1.2.5.8 Примечания

WriteNum и ReadNum, должны использовать следующие алгоритмы кодирования для преобразования во внешний формат и обратно.

```
PROCEDURE WriteNum (VAR r: Rider; x: LONGINT);
BEGIN
 WHILE (x < - 64) OR (x > 63) DO
 Write(r, CHR(x MOD 128 + 128)); x := x DIV 128
 END;
 Write(r, CHR(x MOD 128))
END WriteNum;

PROCEDURE ReadNum (VAR r: Rider; VAR x: LONGINT);
 VAR s: SHORTINT; ch: CHAR; n: LONGINT;
НАЧАТЬ
 s := 0; n := 0;
 Read(r, ch);
 WHILE ORD(ch) >= 128 DO
 INC(n, ASH(ORD(ch) - 128, s) );
 INC(s, 7);
 Read(r, ch)
 END;
 x := n + ASH(ORD(ch) MOD 64 - ORD(ch) DIV 64 * 64, s)
END ReadNum;
```

Причина указания имени файла в операции New заключается в том, чтобы с самого начала распределить файл на нужном носителе (если операционная система поддерживает несколько носителей).

Операции Read, Write, ReadBytes и WriteBytes требуют существования типа SYSTEM.BYTE со следующими характеристиками:

Если формальный параметр имеет тип SYSTEM.BYTE, то соответствующий фактический параметр может иметь тип CHAR, SHORTINT или SYSTEM.BYTE.

Если параметр формальной переменной имеет тип ARRAY OF SYSTEM.BYTE, то соответствующий фактический параметр может быть любого типа. Обратите внимание, что эта возможность опасна и по своей сути не переносима. Поэтому ее использование должно быть ограничено модулями системного уровня.

1.2.5.9 Происхождение

Этот модуль является частью системы ETH Oberon. Приведенная выше спецификация была предложена X. Мёссенбёком, ETH.

1.2.6 Строки модуля

Модуль Strings предоставляет набор операций над строками (т.е. над строковыми константами и массивами символов, оба из которых содержат символ 0X в качестве терминатора). Все позиции в строках начинаются с 0.

```
ОПРЕДЕЛЕНИЕ Строки;
    PROCEDURE Length (s: ARRAY OF CHAR): INTEGER;
    PROCEDURE Insert (source: ARRAY OF CHAR;
                      pos: INTEGER;
                      VAR dest: ARRAY OF CHAR);
    PROCEDURE Append (extra: ARRAY OF CHAR;
                      VAR dest: ARRAY OF CHAR);
    PROCEDURE Delete (VAR s: ARRAY OF CHAR;
                      pos, n: INTEGER);
    ПРОЦЕДУРА Заменить (источник: ARRAY OF CHAR;
                      pos: INTEGER;
                      VAR dest: ARRAY OF CHAR);
    PROCEDURE Extract (источник: ARRAY OF CHAR;
                      pos, n: INTEGER;
                      VAR dest: ARRAY OF CHAR);
    PROCEDURE Pos    (pattern,s: ARRAY OF CHAR;
                      pos: INTEGER): INTEGER;
    PROCEDURE Cap    (VAR s: ARRAY OF CHAR);
END Strings
```

1.2.6.1 Операции

Length(s) возвращает количество символов в s до первого 0X включительно.

Insert(src, pos, dst) вставляет строку src в строку dst в позицию pos (0 <=pos<=Length(dst)). Если pos = Length(dst), src добавляется к dst. Если размер dst недостаточно велик, чтобы вместить результат операции, результат усекается, так что dst всегда завершается символом 0X.

Append(s,dst) имеет тот же эффект, что и Insert(s,Length(dst),dst).

Delete(s, pos, n) удаляет из s n символов, начиная с позиции pos (0 <= pos Length(s)). Если n > Length(s) - pos, то новая длина s равна pos.

Replace(src, pos, dst) имеет тот же эффект, что и Delete(dst, pos, Length(src)) с последующей Insert(src, pos, dst).

Extract(src, pos, n, dst) извлекает подстроку dst с n символами из позиции pos (0 <= pos Length(src)) в src. Если n > Length(src) - pos, то dst - это только часть src от pos до

конец src, т.е. Length(src) -1. Если размер dst недостаточно велик, чтобы вместить результат операции, результат усекается, так что dst всегда завершается символом 0X.

Pos(pat, s, pos) возвращает позицию первого вхождения pat в s. Поиск начинается с позиции pos. Если pat не найден, возвращается -1.

Cap(s) заменяет каждую строчную букву в пределах s на ее эквивалент в верхнем регистре.

1.2.6.2 Примечания

Присваивание и сравнение строк уже поддерживаются языком Oberon- 2.

1.2.6.3 Происхождение

Этот модуль в значительной степени основан на библиотеке ISO Modula-2 Strings, но значительно упрощен. Он был отредактирован Брайаном Кирком, Ником Уолшем, Йозефом Темплом и Ханспетером Мёссенбёком.

1.2.7 Модуль Математика и MathL

Модуль Math предоставляет базовый набор функций общего назначения, использующих арифметику REAL. Модуль MathL предоставляет те же функции для арифметики LONGREAL.

ОПРЕДЕЛЕНИЕ Math;

```
CONST
pi = 3.14159265358979323846;
e = 2.71828182845904523536;
PROCEDURE sqrt (x : REAL) : REAL;
PROCEDURE power (x,base : REAL) : REAL;
PROCEDURE exp (x : REAL): REAL;
PROCEDURE ln (x : REAL) : REAL;
PROCEDURE log (x,base : REAL) : REAL;
PROCEDURE round (x : REAL) : REAL;
PROCEDURE sin (x : REAL) : REAL;
PROCEDURE cos (x : REAL) : REAL
PROCEDURE tan (x : REAL) : REAL;
PROCEDURE arcsin (x : REAL) : REAL;
PROCEDURE arccos (x : REAL) : REAL;
PROCEDURE arctan (x : REAL) : REAL;
PROCEDURE arctan2(x,y : REAL): REAL
PROCEDURE sinh (x:REAL):REAL;
PROCEDURE cosh (x:REAL):REAL;
```

```
PROCEDURE tanh (x:REAL):REAL;
PROCEDURE arcsinh(x:REAL):REAL;
```

```
PROCEDURE arccosh(x:REAL):REAL;
PROCEDURE arctanh(x:REAL):REAL;
```
Математика `END.`

1.2.7.1 Операции

sqrt (x)возвращает квадратный корень из x, где x должно быть положительным

sin, cos, tan (x)возвращает значение синуса, косинуса или тангенса x, где x - в радианах

arcsin, arcos, arctan (x) возвращает значение arcsine, arcos, arctan в радианах от x, где x - это значение синуса, косинуса или тангенса.

power(x, base)возвращает x в базу мощности

round(x) если дробная часть x находится в диапазоне от 0,0 до 0,5, то результатом будет наибольшее целое число, не превышающее x, иначе результатом будет x, округленное до следующего наибольшего целого числа. Обратите внимание, что целочисленные значения не всегда могут быть точно представлены в формате REAL или LONGREAL.

ln(x) возвращает натуральный логарифм (основание e) от x

exp(x)- экспонента от x по основанию e. x не должен быть ни настолько мал, чтобы эта экспонента занижалась, ни настолько велик, чтобы она завышалась.

log(x,base)- логарифм x по основанию b. Допускаются все положительные аргументы. Основание b должно быть положительным.

arctan2(xn,xd)- это тангенс дуги с поправкой на квадрант atan(xn/xd). Если знаменатель xd равен нулю, то числитель xn не должен быть нулем. Все аргументы допустимы, кроме xn = xd = 0.

sinh(x) - гиперболический синус от x. Аргумент x не должен быть настолько большим, чтобы exp(|x|) перетекала.

cosh(x) - гиперболический косинус x. Аргумент x не должен быть настолько большим, чтобы exp(|x|) переполнялся.

tanh(x) - гиперболический тангенс x. Все аргументы являются

допустимыми. arcsinh(x) - дуговой гиперболический синус x.

Все аргументы являются допустимыми.

arccosh(x) - гиперболический косинус дуги x. Все аргументы больше или равны 1.

arctanh(x) - дуговой гиперболический тангенс к x.

|x| < 1 - sqrt(em), где em - машинный эпсилон.

Обратите внимание, что |x| не должен быть настолько близок к 1, чтобы результат был менее точным, чем половинная точность.

1.2.7.2 Источник:

Основан на оригинальном модуле ETH Math, с дополнениями от BK и Эла Фрида, NASA.

<< BK должен ли результат раунда быть LONGINT или LONGREAL?

<< AF round (LONGREAL) в любом случае будет иметь проблемы с точностью.

1.2.8 Модуль Coroutines

Модуль Coroutines предоставляет невытесняющие потоки, каждый из которых имеет свой собственный стек, но при этом разделяет общее адресное пространство. Coroutines может явно передавать управление другим coroutines, которые затем возобновляются с того места, где они передали управление в последний раз.

ОПРЕДЕЛЕНИЕ

```
 Coroutines; TYPE
 Coroutine = RECORD END;
 Body = PROCEDURE;
 PROCEDURE Init (body: Body; stackSize: LONGINT;
                 VAR cor: Coroutine);
 PROCEDURE Transfer (VAR from, to: Coroutine);
END Coroutines.
```

1.2.8.1 Операции

Init(p, s, c)создает и инициализирует новую корутину с со стеком в s байт и телом, представленным в виде процедуры p. Инициализированная корутина может быть запущена передачей в нее. В этом случае ее выполнение начнется с первой инструкции p. Процедура p никогда не должна возвращаться.

Transfer(f, t) передает управление от выполняющейся в данный момент coroutine к coroutine t. Состояние выполняющейся в данный момент coroutine сохраняется в f. При передаче управления обратно в f позже, f будет перезапущена в сохраненном состоянии.

1.2.8.2 Источник

Предложил профессор Ханспетер Мёссенбёк, ETH.

1.2.9 Модули MathC и MathLC

Модуль MathC предоставляет функции для КОМПЛЕКСНОЙ арифметики. Модуль MathLC предоставляет те же функции для LONGCOMPLEX.

ОПРЕДЕЛЕНИЕ `MathC;`

```
PROCEDURE abs (z:COMPLEX):REAL;
PROCEDURE power(z:COMPLEX;base:REAL):COMPLEX;
```

```
 PROCEDURE conj(z:COMPLEX):COMPLEX;
 PROCEDURE sqrt(z:COMPLEX):COMPLEX;
 PROCEDURE exp (z:COMPLEX):COMPLEX;
 PROCEDURE ln (z:COMPLEX):COMPLEX;
 PROCEDURE log (z:COMPLEX,
 b:REAL):КОМПЛЕКС; PROCEDURE sin
 (z:КОМПЛЕКС):КОМПЛЕКС; PROCEDURE cos
 (z:КОМПЛЕКС):КОМПЛЕКС; PROCEDURE tan
 (z:КОМПЛЕКС):КОМПЛЕКС; PROCEDURE arcsin
 (z:КОМПЛЕКС):КОМПЛЕКС; PROCEDURE arccos
 (z:COMPLEX):COMPLEX; PROCEDURE arctan
 (z:COMPLEX):COMPLEX; PROCEDURE arctan2
 (zn,zd:COMPLEX):COMPLEX; PROCEDURE sinh
 (z:COMPLEX):COMPLEX; PROCEDURE cosh
 (z:COMPLEX):КОМПЛЕКС; ПРОЦЕДУРА tanh
 (z:КОМПЛЕКС):КОМПЛЕКС; ПРОЦЕДУРА
 arcsinh (z:КОМПЛЕКС):КОМПЛЕКС;
 ПРОЦЕДУРА arccosh
 (z:КОМПЛЕКС):КОМПЛЕКС; ПРОЦЕДУРА
 arctanh (z:КОМПЛЕКС):КОМПЛЕКС;
END MathC.
```

1.2.9.1 Операции

z = x + iy

bnis наибольшее число с плавающей запятой для данной машины. emis машинный эпсилон.

esis em, деленное на арифметическое основание машины.

sn - наименьшее число с плавающей запятой для данной машины.

abs(z)- абсолютное значение или величина комплексного числа z. Аргументы x и y не должны быть настолько большими, чтобы x*x + y*y переполнялись. Возвращаемое значение - вещественное число.

power (Z,base) возвращает Z в базис мощности, см.

комментарии для exp. conj(z) - комплексная сопряженность z.

Все аргументы являются законными.

sqrt(z) - комплексный квадратный корень из z. Абсолютное значение z не должно переполняться.

exp(z) - это комплексная экспонента от z в базисе e. Вещественная часть z, т.е. x, не должна быть ни настолько мала, чтобы результат занижался, ни настолько велика, чтобы переполнялся. Если |y| слишком велико, результат может быть менее точным, чем половинная точность. Если |y| очень велико, результат не будет иметь точности.

ln(z) - комплексный натуральный логарифм (основание e) от z. Аргумент не должен быть нулем, а абсолютное значение z не должно переполняться.

log(z,b) - комплексный натуральный логарифм от z по основанию b. Аргумент не должен быть нулем, а абсолютное значение z не должно переполняться. Основание b должно быть положительным.

sin(z) - комплексный синус z.

|Re(z)| = |x| <= 1/sqrt(em) = x(warn)

|Re(z)| = |x| <= 1/em = x(max)

|Im(z)| = |y| <= ln(bn) = y(max)

Если |x| больше, чем x(warn), то результат будет иметь точность меньше половины. Если |x| больше x(max), то результат не будет иметь точности. Наконец, если |y| слишком велико, результат будет переполнен.

cos(z) - комплексный косинус от z.

|Re(z)| = |x| <= 1/sqrt(em) = x(warn)

|Re(z)| = |x| <= 1/em = x(max)

|Im(z)| = |y| <= ln(bn) = y(max)

Если |x| больше, чем x(warn), то результат будет иметь точность меньше половины. Если |x| больше x(max), то результат не будет иметь точности. Наконец, если |y| слишком велико, результат будет переполнен.

tan(z) - это комплексный тангенс z. Если |cos(z)|**2 очень мал, то есть если x очень близок к pi/2 или 3*pi/2, а y мал, то tan(z) будет почти единичным. Если |cos(z)|**2 несколько больше, но все еще мал, то результат будет менее точным, чем половинная точность. Когда 2x настолько велико, что sin(2x) не может быть вычислен с любой ненулевой точностью, возникает особая ситуация.
Если |y| < 3/2, то tan не может быть оценен

с точностью лучше, чем одна значащая цифра. Если 3/2 <= |y| < -0,5*ln(es/2), то tan можно вычислить, игнорируя действительную часть аргумента; однако ответ будет менее точным, чем с половинной точностью.

arcsin(z) - комплексный синус дуги z. |x| должен быть меньше или

равен 1. arccos(z) - комплексный косинус дуги z. |x| должен быть

меньше или равен 1.

arctan(z) - комплексный тангенс дуги к z. Аргумент z не должен быть в точности +/-i, так как atan(+/-i) не определен. Кроме того, z не должно быть настолько близко к +/-i, чтобы потерять существенное значение.

arctan2(zn,zd) - квадратно-корректный комплексный дуговой тангенс atan(zn/zd).

Отношение z = zn/ zd не должно быть +/-i, потому что atan(+/-i) не определено. Аналогично, zn и zd не должны быть равны нулю. Наконец, z не должно быть настолько близко к +/-i, чтобы потерять существенное значение.

sinh(z) - гиперболический синус z.

|Im(z)| = |y| <= 1/sqrt(em) = y(warn)

|Im(z)| = |y| <= 1/em = y(max)

|Re(z)| = |x| <= ln(bn) = x(max)

Если |y| больше, чем y(warn), то результат будет менее точным, чем половина точности. Если |y| больше y(max), то результат не будет иметь точности.

Наконец, если |x| слишком велик, результат

переполняется. cosh(z) - гиперболический

косинус z.

|Im(z)| = |y| <= 1/sqrt(em) = y(warn)|Im(z)| = |y| <= 1/em = y(max)

|Re(z)| = |x| <= ln(bn) = x(max)

Если |y| больше, чем y(warn), то результат будет менее точным, чем половина точности. Если |y| больше y(max), то результат не будет иметь точности. Наконец, если |x| слишком велико, результат будет переполнен.

tanh(z) - это гиперболический тангенс к z. Если |cosh(z)|**2 очень мал, то есть если y mod 2*pi очень близко к pi/2 или 3*pi/2 и если x мал, то tanh(z) будет почти сингулярным. Если |cosh(z)|**2 несколько больше, но все равно мал, то результат будет меньше

точнее, чем половинная точность. Когда 2y настолько велико, что sin(2y) не может быть вычислено точно даже с нулевой точностью, возникает особая ситуация. Если |x| < 3/2, то tanh не может быть вычислен с точностью лучше единицы.

значащая цифра. Если 3/2 <=|y| < - 0,5*ln(es/2), то tanh можно вычислить, игнорируя мнимую часть аргумента; однако ответ будет менее точным, чем половина прецизионного значения.

arcsinh(z) - дуговой гиперболический синус z. Почти все аргументы являются допустимыми. Только когда |z| > bn/2, может произойти переполнение.

arccosh(z)- гиперболический косинус дуги для z. Почти все аргументы являются допустимыми. Только когда |z|
> bn/2 может произойти переполнение.

arctanh(z) - это дуговой гиперболический тангенс к z. Аргумент не должен быть точно +/-1, потому что дуговой гиперболический тангенс к z там не определен. Кроме того, z не должно быть настолько близко к +/-1, чтобы потерять существенное значение.

1.2.9.2 Источник

Предложено Элом Фридом, NASA. Основано на пакете IMSL.

Приложение B: Список авторов

Разработчики	**компилятораЭлектронный адрес**
Эндрю Кадах	71333.2346@compuserv.com
Пол Кертис	
Гюнтер Дотцель	100023.2527@compuserv.com
Джон Гоф	gough@fitmail.fit.qut.edu.au
Тейлор Хатт	thutt@access.digex.com
Брайан	
	Kirkrobinsons@cix.compulink.c
o.uk Ханспетер	
	MössenböckMoessenboeck@cs.
inf.ethz.ch Алекс	Nedoryaned@isi.itfs.nsk.su
Куно Пфистер	pfister@inf.ethz.ch
Йозеф Темпл	templ@inf.ethz.ch
Рик Уотсон	watson@futurs.enet.dec.com

Рецензенты заявок

Стив Коллинз	71333.2346@compuserve.com
Эл Фрид	al@sarah.lerc.nasa.gov
Юэн Хилл	100143.1660@compuserve.com
Стив Метцелер----	
Аня Шумахер-	
Стив Террапин	100023.1307@compuserve.com
Ник Уолш (покойный)	-.. .

Приложение C: Конференция в Оквуде

Кройдон 21 ... 23 июня 1993 г.

3.1 Список авторов и участников

Имя (инициалы)	Страна	Организация
Кристоф Брасс (CB)	Швейцария	Аналитик АГ
Оливер Бройнингер (OB)	Германия	Индивидуалка
Эндрю Кадах (AC)	Россия	ИСИ СД РАН
Стив Коллинз (SC)	ВЕЛИКОБРИТАНИЯ	Real Time Assoc.
Андреас Дистели (AD)	Швейцария	ETH
Гюнтер Дотцель (GD)	Германия	ModulaWare GmbH
Дэйв Фокс (DF)	ВЕЛИКОБРИТАНИЯ	Real Time Assoc.
Джон Гоф (JG)	Австралия	QUT Карден Пойнт
Джим Хокинс (JH)	ВЕЛИКОБРИТАНИЯ	Amiga
Юэн Хилл (EH)	ВЕЛИКОБРИТАНИЯ	BSC SIG
Стиг Холмберг (SH)	Швеция	Остерсундский университет.
Вольфганг Хугентобер (WH)	Швейцария	Л Кисслинг и Ко. AG
Тейлор Хатт (TH)	США	Индивидуалка
Брайан Кирк (BK)	ВЕЛИКОБРИТАНИЯ	Robinson Assoc.
Ханс Клявер (HK)	Нидерланды	Индивидуалка
Бернхард Лейш (BL)	Австрия	Иоганн Кеплер
Стив Метцелер (SM)	Швейцария	Alchemia Software
Алекс Недория (AN)	Россия	ИСИ СД РАН
Куно Пфистер (CP)	Швейцария	Oberon Microsys.
Маркус Раубер (MR)	Швейцария	CATS AG
Стив Рамсби (SR)	ВЕЛИКОБРИТАНИЯ	Де Монтфорд
Петер Шультес (PS)	Германия	Ульмский университет
Аня Шумахер (AS)	Германия	Сименс АГ
Фритьоф Зиберт (FS)	Германия	Программное обеспечение для Amiga
Йозеф Темпл (JT)	Швейцария	ETH
Стив Терепин (ST)	ВЕЛИКОБРИ	Opus 1 Software

	ТАНИЯ	
Ник Уолш (NJW)	ВЕЛИКОБРИТАНИЯ	Городской университет
Рик Уотсон (RW)	ВЕЛИКОБРИТАНИЯ	DEC

Материалы предварительной конференции и/или извинения были получены от следующих лиц, которые не смогли присутствовать на конференции ...

Уитни де Врис (WV)	Канада	Университет Макгилла
J Gutknecht (JG)	Швейцария	ETH
Шерил Линс (CL)	США	Apple Corp.

Ян Маршалл (IM)	ВЕЛИКОБРИТАНИЯ	Real Time Assoc.
Майкл МакГоу (MM)	США	НАСА
Ханспетер Мёссенбёк (HM)	Швейцария	ETH
Алан Фрид (AF)	США	НАСА
Крис Джонсон (CJ)	США	НАСА
Никлаус Вирт (NW)	Швейцария	ETH
Дик Паунтин (ДП)	ВЕЛИКОБРИТАНИЯ	Журнал BYTE
Марк Вудман (MW)	ВЕЛИКОБРИТАНИЯ	Открытый университет

3.2 Изменение документа Запись

Пересмотр	**Описание**	**Дата**	**Имя**
0A	Первоначальная версия Отправлено по электронной почте для комментариев	Июнь 93 года	BK
0B	Сильно переработанная версия, основанная на отзывы от авторов, перечисленных в Приложение B	Oct. 93	БК и др.
0C	Черновик рецензирован Й. Темплом, Х. Мёссенбёком, октябрь 93 г. Б Кирк		BK/ETH
0D	Поправки и уточнения от JT, г. Группа HM и NASA под редакцией Предлагаемая первая эмиссия до ETH предисловие	Ноябрь 93	БК и др.
1A	Окончательная редакция и добавление предисловияДекабрь 93 г	.	BK/HM

3.3 Документ Обратная связь

Мы хотели бы получить от вас комментарии по поводу этого документа или предложения по его улучшению. Пожалуйста, присылайте ваши комментарии по адресу

Oakwood Guidelines
Robinson Associates
Red Lion House
Сент-Мэри-стрит
Пейнсвик GLOS GL6
6QR
Голос (+ 44) (0)452 813 699
Факс (+ 44) (0)452 812 912
e-Mail: robinsons@cix.compulink.co.uk

Будет очень полезно, если вы укажете конкретные ссылки на текст, где это необходимо, и, конечно, ваше собственное имя и адрес позволят ответить на ваши комментарии.

Имя:

Адрес:

Страна:Электронная почта:

Телефон:Факс:

Название:

Выпуск:

Комментарии и предложения (при необходимости добавьте дополнительные страницы) :