Компилятор языка программирования Oberon-07/11

Данная реализация является демонстрационной: предоставляет разработчику лишь минимальные удобства; имеет сильно ограниченную стандартную библиотеку; оптимизация отсутствует.

Состав программы

1. Editor.exe – многостраничный текстовый редактор для редактирования файлов модулей (*.ob07) с подсветкой синтаксиса, нумерацией строк и автоматическим преобразованием ключевых слов и стандартных идентификаторов к верхнему регистру
2. Compiler.exe – транслятор исходного кода (*.ob07) в машинные коды x86 (формат исполняемых файлов – PE (*.exe и *.dll)), написан на Oberon-07/11 (самотранслятор). Параметры:
   1) полное имя главного модуля
   2) тип приложения “con”, “gui” или “dll”
   3) размер стека в мегабайтах

   Например: "C:\oberon-07\example.ob07" con 1

   В случае успешной компиляции, компилятор передает код завершения 0, иначе 1
3. Compile.cmd – командный файл для запуска компилятора из редактора
4. папка std – содержит стандартные модули

Отличия от оригинала.

1. Исключен сборщик мусора, для компенсации введена стандартная процедура `DISPOSE(v: тип-указатель)`, процедура освобождает память и присваивает переменной `v` значение `NIL`.
2. Расширен модуль `SYSTEM`.
3. В алфавит добавлен символ подчеркивания `"_"`, он может использоваться в идентификаторах.
4. Добавлены флаги соглашения вызова.

Особенности реализации.

1. Основные типы.

Тип	Диапазон значений	Размер, байт
INTEGER	-2147483648 .. 2147483647	4
REAL	1.40E-45 .. 3.34E+38	4
LONGREAL	4.94E-324 .. 1.70E+308	8
CHAR	символ ASCII (0X .. 0FFX)	1
BOOLEAN	FALSE, TRUE	1
SET	множество из целых чисел {0 .. 31}	4

2. Максимальная длина идентификаторов – 255 символов
3. Максимальная длина строковых констант – 255 символов
4. Максимальная длина строк исходного кода – 1024 символа
5. Максимальная размерность открытых массивов – 5
6. Максимальное количество объявленных типов-записей – 2047
7. Стандартная процедура NEW при выделении блока памяти, заполняет его нулями
8. Глобальные и локальные переменные инициализируются нулями

Модуль SYSTEM

Модуль SYSTEM содержит низкоуровневые и небезопасные процедуры, ошибки при использовании процедур модуля SYSTEM могут привести к повреждению данных времени выполнения и аварийному завершению программы.

```
PROCEDURE ADR(v: любой тип): INTEGER
     v – переменная, процедура или строковая константа
     результат - адрес v

PROCEDURE BIT(a, n: INTEGER): BOOLEAN
     n-й бит Память[a] (0 <= n <= 7)

PROCEDURE SIZE(T): INTEGER
     размер типа T

PROCEDURE TYPEID(T): INTEGER
     номер типа в таблице типов-записей
     T – тип-запись или тип-указатель

PROCEDURE GET(a: INTEGER; VAR v: любой основной тип)
     v := Память[a]

PROCEDURE PUT(a: INTEGER; x: любой основной тип)
     Память[a] := x

PROCEDURE CODE(s: ARRAY OF CHAR)
     Вставка машинного кода
     s – строковая константа шестнадцатиричных цифр
     количество цифр должно быть четным
     например: SYSTEM.CODE("B801000000") (* mov eax, 1 *)
```

```
PROCEDURE LOADLIB(name: INTEGER): INTEGER
     Загрузка dll-библиотеки, name – имя файла (адрес строки),
     возвращает числовой идентификатор библиотеки (в случае ошибки
     возвращает 0)

PROCEDURE FREELIB(lib: INTEGER): BOOLEAN
     Выгрузка загруженной ранее dll-библиотеки с числовым
     идентификатором lib

PROCEDURE GETPROC(lib, name: INTEGER): INTEGER
     Возвращает адрес процедуры из dll-библиотеки с числовым
     идентификатором lib, name – имя процедуры (адрес строки). В случае
     ошибки возвращает 0

PROCEDURE HALT(code: INTEGER)
     Завершает приложение и передает code во внешнюю среду
```

Также в модуле SYSTEM определен дополнительный основной тип CARD16 (2 байта). Для типа CARD16 не допускаются никакие явные операции, за исключением присваивания. Преобразования CARD16 → INTEGER и INTEGER → CARD16 могут быть реализованы так:

```
PROCEDURE Card16ToInt(w: SYSTEM.CARD16): INTEGER;
VAR i: INTEGER;
BEGIN
  SYSTEM.PUT(SYSTEM.ADR(i), w)
  RETURN i
END Card16ToInt;

PROCEDURE IntToCard16(i: INTEGER): SYSTEM.CARD16;
VAR w: SYSTEM.CARD16;
BEGIN
  SYSTEM.GET(SYSTEM.ADR(i), w)
  RETURN w
```

```
END IntToCard16;
```

Флаги соглашения вызова

При объявлении процедурных типов и глобальных процедур, после ключевого слова `PROCEDURE` может быть указан флаг соглашения вызова.
Его формат: `"[" ( "stdcall" | "cdecl" ) "]"`, то есть `[stdcall]` или `[cdecl]`.
Например:

```
PROCEDURE [cdecl] MyProc(x, y, z: INTEGER): INTEGER;
```

По умолчанию принимается соглашение `stdcall`.

Скрытые параметры процедур.

Некоторые процедуры могут иметь скрытые параметры, они отсутствуют в списке формальных параметров, но используются компилятором. Это возможно в следующих случаях:

1. Процедура имеет формальный параметр открытый массив:
   `PROCEDURE Proc(x: ARRAY OF ARRAY OF LONGREAL);`
   Вызов транслируется так:
   `Proc(SYSTEM.ADR(x), LEN(x), LEN(x[0])`
2. Процедура имеет формальный параметр-переменную типа `RECORD`:
   `PROCEDURE Proc(VAR x: Rec);`
   Вызов транслируется так:
   `Proc(SYSTEM.TYPEID(Rec), SYSTEM.ADR(x))`
3. Процедура является вложенной, глубина вложения `k`, для глобальных процедур `k = 0`
   `PROCEDURE Proc(p1, p2, … pn);`
   Вызов транслируется так:
   `Proc(base(k - 1), base(k - 2), … base(0), p1, p2, … pn)`, где `base(m)` – адрес базы кадра стека охватывающей процедуры глубины вложения m (используется для доступа к локальным переменным охватывающей процедуры)

Модуль RTL

Модуль RTL обязательно должен импортироваться всеми программами. Компилятор транслирует некоторые операции (проверка/охрана типа, сравнение строк, сообщения об ошибках времени выполнения и др.) как вызовы процедур модуля RTL. Пользователь не должен вызывать явно процедуры этого модуля за исключением процедуры `SetClose`:

```
PROCEDURE SetClose(proc: PROC), где TYPE PROC = PROCEDURE
```

`SetClose` назначает процедуру `proc` (без параметров) вызываемой при выгрузке dll-библиотеки, если приложение компилируется как dll. Для exe-приложений вызов процедуры `SetClose` ни на что не влияет.

стандартные модули

`MODULE Out` - консольный вывод

`PROCEDURE Open`

открывает консольный вывод

`PROCEDURE Int(x, width: INTEGER)`

вывод целого числа `x`;

`width` - количество знакомест, используемых для вывода

`PROCEDURE Real(x: LONGREAL; width: INTEGER)`

вывод вещественного числа `x` в плавающем формате;

`width` - количество знакомест, используемых для вывода

`PROCEDURE Char(x: CHAR)`

вывод символа `x`

`PROCEDURE FixReal(x: LONGREAL; width, p: INTEGER)`

вывод вещественного числа `x` в фиксированном формате;

`width` - количество знакомест, используемых для вывода;

`p` - количество знаков после десятичной точки

`PROCEDURE Ln`

переход на следующую строку

`PROCEDURE String(s: ARRAY OF CHAR)`

вывод строки `s`

`MODULE In` - консольный ввод

`VAR Done: BOOLEAN`

принимает значение `TRUE` в случае успешного выполнения операции ввода и `FALSE` в противном случае

`PROCEDURE Open`

открывает консольный ввод,
также присваивает переменной `Done` значение `TRUE`

`PROCEDURE Int(VAR x: INTEGER)`

ввод числа типа `INTEGER`

`PROCEDURE Char(VAR x: CHAR)`

ввод символа

`PROCEDURE Real(VAR x: REAL)`

ввод числа типа `REAL`

`PROCEDURE LongReal(VAR x: LONGREAL)`

ввод числа типа `LONGREAL`

`PROCEDURE String(VAR s: ARRAY OF CHAR)`

ввод строки

`PROCEDURE Ln`

ожидание нажатия `ENTER`

`MODULE Console` – дополнительные процедуры консольного вывода

`CONST`

Следующие константы определяют цвет консольного вывода

```
Black = 0 Blue = 1 Green = 2 Cyan = 3 Red = 4 Magenta = 5
Brown = 6 LightGray = 7 DarkGray = 8 LightBlue = 9
LightGreen = 10 LightCyan = 11 LightRed = 12 LightMagenta = 13
Yellow = 14 White = 15
```

`PROCEDURE Cls`

очистка окна консоли

`PROCEDURE Color(FColor, BColor: INTEGER)`

установка цвета консольного вывода: `FColor` – цвет текста,

`BColor` – цвет фона, возможные значения – вышеперечисленные константы

`PROCEDURE Cursor(x, y: INTEGER)`

установка курсора консоли в позицию (`x`, `y`)

`PROCEDURE CursorX(): INTEGER`

возвращает текущую x-координату курсора консоли

`PROCEDURE CursorY(): INTEGER`

возвращает текущую y-координату курсора консоли

`MODULE Math` - математические функции

```
CONST pi = 3.14159265358979323846264338327950D+00
```

```
VAR INF, negINF: LONGREAL
```

положительная и отрицательная бесконечность

```
PROCEDURE IsNan(x: LONGREAL): BOOLEAN
```

возвращает `TRUE`, если `x` – не число

```
PROCEDURE IsInf(x: LONGREAL): BOOLEAN
```

возвращает `TRUE`, если `x` – бесконечность

```
PROCEDURE sqrt(x: LONGREAL): LONGREAL
```

квадратный корень `x`

```
PROCEDURE exp(x: LONGREAL): LONGREAL
```

экспонента `x`

```
PROCEDURE ln(x: LONGREAL): LONGREAL
```

натуральный логарифм `x`

```
PROCEDURE sin(x: LONGREAL): LONGREAL
```

синус `x`

```
PROCEDURE cos(x: LONGREAL): LONGREAL
```

косинус `x`

```
PROCEDURE tan(x: LONGREAL): LONGREAL
```

тангенс `x`

```
PROCEDURE arcsin(x: LONGREAL): LONGREAL
```

арксинус `x`

```
PROCEDURE arccos(x: LONGREAL): LONGREAL
```

арккосинус `x`

```
PROCEDURE arctan(x: LONGREAL): LONGREAL
```

арктангенс `x`

```
PROCEDURE arctan2(y, x: LONGREAL): LONGREAL
```

арктангенс `y/x`

```
PROCEDURE power(base, exponent: LONGREAL): LONGREAL
```

возведение числа `base` в степень `exponent`

```
PROCEDURE log(base, x: LONGREAL): LONGREAL
```

логарифм `x` по основанию `base`

```
PROCEDURE sinh(x: LONGREAL): LONGREAL
```

гиперболический синус `x`

```
PROCEDURE cosh(x: LONGREAL): LONGREAL
```

гиперболический косинус `x`

```
PROCEDURE tanh(x: LONGREAL): LONGREAL
```

гиперболический тангенс `x`

```
PROCEDURE arcsinh(x: LONGREAL): LONGREAL
```

обратный гиперболический синус `x`

```
PROCEDURE arccosh(x: LONGREAL): LONGREAL
```

обратный гиперболический косинус `x`

```
PROCEDURE arctanh(x: LONGREAL): LONGREAL
```

обратный гиперболический тангенс `x`

```
PROCEDURE round(x: LONGREAL): LONGREAL
```

округление `x` до ближайшего целого

```
PROCEDURE frac(x: LONGREAL): LONGREAL;
```

дробная часть числа `x`

```
PROCEDURE floor(x: LONGREAL): LONGREAL
```

наибольшее целое число (представление как `LONGREAL`),
не больше `x`: `floor(1.2) = 1.0`

```
PROCEDURE ceil(x: LONGREAL): LONGREAL
```

наименьшее целое число (представление как `LONGREAL`),
не меньше `x`: `ceil(1.2) = 2.0`

```
PROCEDURE sgn(x: LONGREAL): INTEGER
```

если `x > 0` возвращает `1`
если `x < 0` возвращает `-1`
если `x = 0` возвращает `0`

MODULE File - файловый двоичный ввод-вывод

```
CONST

OPEN_R = 0
OPEN_W = 1
OPEN_RW = 2

SEEK_BEG = 0
SEEK_CUR = 1
SEEK_END = 2
```

```
PROCEDURE Create(FName: ARRAY OF CHAR): INTEGER
```

создает новый файл с именем FName (полное имя с путем), открывет файл для записи и возвращает идентификатор файла (целое число), в случае ошибки, возвращает -1

```
PROCEDURE Open(FName: ARRAY OF CHAR; Mode: INTEGER): INTEGER
```

открывает существующий файл с именем FName (полное имя с путем) в режиме Mode = (OPEN_R (только чтение), OPEN_W (только запись), OPEN_RW (чтение и запись)), возвращает идентификатор файла (целое число), в случае ошибки, возвращает -1

```
PROCEDURE Read(F, Buffer, Count: INTEGER): INTEGER
```

Читает данные из файла в память

F - числовой идентификатор файла, Buffer – адрес области памяти, Count – количество байт, которое требуется прочитать из файла;

возвращает количество байт, которое было прочитано из файла

```
PROCEDURE Write(F, Buffer, Count: INTEGER): INTEGER
```

Записывает данные из памяти в файл

F - числовой идентификатор файла, Buffer – адрес области памяти, Count – количество байт, которое требуется записать в файл;

возвращает количество байт, которое было записано в файл

`PROCEDURE Seek(F, Offset, Origin: INTEGER): INTEGER`

устанавливает позицию чтения-записи файла с идентификатором `F` на `Offset`, относительно `Origin` = (`SEEK_BEG` – начало файла, `SEEK_CUR` – текущая позиция, `SEEK_END` – конец файла), возвращает позицию относительно начала файла, например: `Seek(F, 0, 2)` - устанавливает позицию на конец файла и возвращает длину файла; при ошибке возвращает -1

`PROCEDURE Close(F: INTEGER)`

закрывает ранее открытый файл с идентификатором `F`

`PROCEDURE Delete(FName: ARRAY OF CHAR): BOOLEAN`

удаляет файл с именем `FName` (полное имя с путем), возвращает `TRUE`, если файл успешно удален

`PROCEDURE Exists(FName: ARRAY OF CHAR): BOOLEAN`

возвращает `TRUE`, если файл с именем `FName` (полное имя) существует

`MODULE Read` – чтение основных типов данных из файла `F`

Процедуры возвращают `TRUE` в случае успешной операции чтения

```
PROCEDURE Char(F: INTEGER; VAR x: CHAR): BOOLEAN

PROCEDURE Int(F: INTEGER; VAR x: INTEGER): BOOLEAN

PROCEDURE Real(F: INTEGER; VAR x: REAL): BOOLEAN

PROCEDURE LongReal(F: INTEGER; VAR x: LONGREAL): BOOLEAN

PROCEDURE Boolean(F: INTEGER; VAR x: BOOLEAN): BOOLEAN
```

```
PROCEDURE Set(F: INTEGER; VAR x: SET): BOOLEAN
```

`MODULE Write` – запись основных типов данных в файл `F`

Процедуры возвращают `TRUE` в случае успешной операции записи

```
PROCEDURE Char(F: INTEGER; x: CHAR): BOOLEAN

PROCEDURE Int(F: INTEGER; x: INTEGER): BOOLEAN

PROCEDURE Real(F: INTEGER; x: REAL): BOOLEAN

PROCEDURE LongReal(F: INTEGER; x: LONGREAL): BOOLEAN

PROCEDURE Boolean(F: INTEGER; x: BOOLEAN): BOOLEAN

PROCEDURE Set(F: INTEGER; x: SET): BOOLEAN
```

`MODULE Dir` – работа с папками

```
PROCEDURE Create(DirName: ARRAY OF CHAR): BOOLEAN
```

создает папку с именем `DirName`, все промежуточные папки должны существовать

```
PROCEDURE Remove(DirName: ARRAY OF CHAR): BOOLEAN
```

удаляет пустую папку с именем `DirName`

```
PROCEDURE Exists(DirName: ARRAY OF CHAR): BOOLEAN
```

возвращает `TRUE`, если папка с именем `DirName` существует

MODULE DateTime – дата, время

```
CONST ERR = -7.0D5
```

```
PROCEDURE Now(): LONGREAL
```

возвращает текущую системную дату и время

```
PROCEDURE EncodeDate(Year, Month, Day, Hour, Min,
    Sec, MSec: INTEGER): LONGREAL
```

возвращает дату, полученную из компонентов
Year, Month, Day, Hour, Min, Sec, MSec;
при ошибке возвращает константу `ERR = -7.0D5`

```
PROCEDURE DecodeDate(Date: LONGREAL; VAR Year, Month,
    Day, Hour, Min, Sec, MSec: INTEGER): BOOLEAN
```

извлекает компоненты
Year, Month, Day, Hour, Min, Sec, MSec из даты Date;
при ошибке возвращает FALSE

MODULE Utils - разное

```
VAR ParamCount: INTEGER
```

- количество параметров программы

```
PROCEDURE AnsiToOem(VAR str: ARRAY OF CHAR)
```

преобразует символы строки str из кодировки Windows-1251 в кодировку CP-866 для вывода кириллицы на консоль

```
PROCEDURE ParamStr(VAR str: ARRAY OF CHAR; n: INTEGER)
```

записывает в строку str n-й параметр программы

MODULE Rnd – генератор случайных чисел

```
PROCEDURE PutSeed(seed: INTEGER)
```

Инициализация генератора целым числом

```
PROCEDURE RND(range : INTEGER): INTEGER
```

Целые случайные числа в диапазоне `0 <= x < range.`

`Range` должен быть в интервале `1 .. 2^31-2`

```
PROCEDURE Random(): LONGREAL
```

Вещественные случайные числа в диапазоне `0.0 <= x < 1.0`

MODULE Windows – процедуры создания окна Windows

```
PROCEDURE CreateWindow(lpClassName, lpWindowName, dwStyle,
     X, Y, nWidth, nHeight, hWndParent, hMenu, hInstance,
     lpParam: INTEGER): INTEGER

PROCEDURE CreateWindowEx(dwExStyle, lpClassName, lpWindowName,
     dwStyle, X, Y, nWidth, nHeight, hWndParent, hMenu, hInstance,
     lpParam: INTEGER): INTEGER
```