## TParserCore. Базовый модуль парсера кода.

Интерфейсные функции и свойства:

* tib: string;

The Input Block.

Строка для парсинга.

* separators: string;

Разделители

* token: string;

Последний выкушенный токен.

* constructor Create; virtual;

Создание модуля.

* function Parse: Boolean; virtual;

Парсинг. Возвращает истину, если удалось выкусить токен.

Идея модуля заключается в том, чтобы обеспечить основные механизмы парсинга и освободить программиста от постоянного написания одних и тех же процедур. По умолчанию, выкусываются токены, разделенные пробелами. Переопределение процедуры Parse позволит реализовать более сложные варианты парсинга, путем изменения разделителей во время работы.

## TCompilerCore. Базовый модуль компилятора.

Цель написания данного модуля – обеспечение базы для написания целевых компиляторов. Модуль состоит из нескольких составных частей:

TParserCore – обеспечение парсинга.

Работа с памятью кода/данных.

Работа со стеком управлений (для if-then-else, for и т.п.)

Работа со словарем токенов

Интерфейсные операции.

Идея модуля – реализация базовых функций компилятора, без которых невозможно обойтись и обеспечение гибкого развития данного модуля.

## Пример использование. Написание компилятора Brainfuck.

В Brainfuck команды односимвольные

|  |  |  |
| --- | --- | --- |
| Команда | Код | Описание |
| + | 0 | Инкремент значения ячейки памяти под указателем |
| - | 1 | Декремент значения ячейки памяти под указателем |
| > | 2 | Сдвиг указателя вправо |
| < | 3 | Сдвиг указателя влево |
| [ | 4 | Если в ячейке под указателем ноль, то переход к соответствующей "]" |
| ] | 5 | Переход к соответствующей "[" |
| , | 6 | Чтение символа с консоли |
| . | 7 | Вывод символа на консоль |

Базовая версия парсера выкусывает команды разделенные пробелами, но это не совсем то, что нам нужно. Переопределяем процедуру Parse.

TBrainfuckParser = class(TParserCore)

public

function Parse: boolean; override;

end;

...

{ TBrainfuckParser }

function TBrainfuckParser.Parse: boolean;

const

ValidChar: string = '+-><[],.';

var

index: integer;

begin

index := 0;

while (FTibPos <= FTibLength) do

begin

index := Pos(Tib[FTibPos], ValidChar);

if index > 0 then

break;

end;

FToken := Tib[FTibPos];

Result := index > 0;

end;

Нам практически не нужны возможности даже базового компилятора, поэтому просто переписываем полностью процедуру Parse. В Token, по прежнему, записывается последний токен, и функция, по прежнему, возвращает успешность парсинга. Меняется только реализация, но не интерфейс функции.

С парсером закончили. Продвигаемся дальше.

Что нам нужно для компиляции кода? Нужно создать 8 процедур для обработки каждого типа токена. И, возможно, изменить процедуру Evaluate. Хотя можно и не менять, переписав TryDispatchNumber, чтобы он ничего не делал, возвращая всегда истину.

TBrainfuckTC = class(TCompilerCore)

protected

procedure TryDispatchNumber; override;

public

Parser: TBrainfuckParser;

end;

...

procedure TBrainfuckTC.TryDispatchNumber;

begin

// делать ничего не нужно. Любой непонятный токен в брейнфаке

// просто пропускается

{do nothing};

end;

Попытка запустить компилятор в таком виде провалилась – он не "подцепил" новый парсер. Немного шевелим мозгами и подменяем парсер.

TBrainfuckTC = class(TCompilerCore)

protected

procedure TryDispatchNumber; override;

public

constructor Create;

end;

...

constructor TBrainfuckTC.Create;

begin

inherited Create;

Parser.Destroy;

Parser := TBrainfuckParser.Create;

end;

Позволяем CompilerCore создать базовый парсер. Сразу же его уничтожаем и создаем на его месте свой с нужной функции. Т.к. он унаследован от TParserCore с одной перекрытой функцией – все замечательно начинает работать.

Дело за малым. Осталось реализовать процедуры обработки токенов.

Только вот наврал я насчет 8 процедур. Достаточно всего одной тривиальной.

TBrainfuckTC = class(TCompilerCore)

private

procedure BrainfuckWord; // <-- добавленная процедура

protected

procedure TryDispatchNumber; override;

public

constructor Create;

procedure InitVocabulary; // <-- добавленная процедура

end;

...

procedure TBrainfuckTC.BrainfuckWord;

begin

Compile(FVocabulary[TokenID].tag);

end;

...

procedure TBrainfuckTC.InitVocabulary;

begin

ClearVocabulary;

AddToken(TokenWord('+', 0, false, BrainfuckWord));

AddToken(TokenWord('-', 1, false, BrainfuckWord));

AddToken(TokenWord('>', 2, false, BrainfuckWord));

AddToken(TokenWord('<', 3, false, BrainfuckWord));

AddToken(TokenWord('[', 4, false, BrainfuckWord));

AddToken(TokenWord(']', 5, false, BrainfuckWord));

AddToken(TokenWord(',', 6, false, BrainfuckWord));

AddToken(TokenWord('.', 7, false, BrainfuckWord));

end;

Инициализация словаря вызывается в процедуре Create.

constructor TBrainfuckTC.Create;

begin

inherited Create;

Parser.Destroy;

Parser := TBrainfuckParser.Create;

InitVocabulary; // <-- добавленная строчка

end;

Вот и все. Компилятор готов. Все рутинные функции обеспечивает базовый класс.