Языки программирования. Лекция 2. Как разрабатывать. Личная история

Алексей Недоря, декабрь 2025

Профессия: Языки программирования и компиляторы

Краткая профессиональная биография:

- Первый компилятор: 1984
- Компиляторы для 9 языков программирования
- Участие в стандартизации языков программирования
  - Modula-2 ISO/IEC
  - Oberon-2 Oakwood Guidelines
- Системная архитектура
- Разработка 7 языков программирования

- t.me/vorchalki_o_prog
- http://digital-economy.ru/avtory/aleksei-nedoria-synergetic-lab-ru
- https://ontonet.org/gruppy/vorchalki-o-programmirovanii
- http://алексейнедоря.рф/

@VORCHALKI_O_PROG

Ликбез: Разработка языков программирования

Прочитано:

- История. Зачем и почему. 26.11.2025 запись

17.12.2025:

- Как разрабатывать. Личная история
  - Модула-0
  - Расширение Модулы-2 до Модулы-X
  - Вир/a0
  - Вир/a1
  - Первый корпоративный (язык К1)
  - Итоги

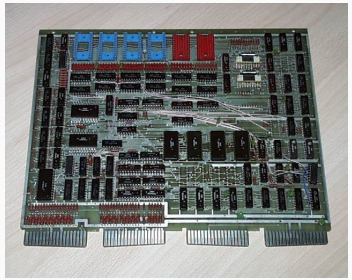
Следующие лекции:

- Как. Общий подход
- Как. Ответы на вопросы, пример Тривиля
- ?

Путь через Модулу-2: от Бахуса (Burroughs 6700) к Кроносу

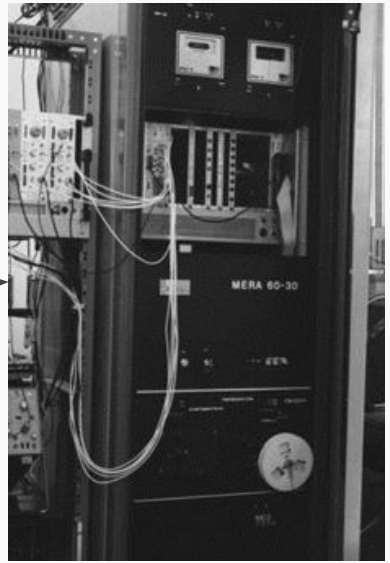
1983-1984

- Leo Кузнецов: Модула-2 компилятор для Кроноса на B6700 (ALGOL Burroughs)
- Leo и я: эмулятор Кроноса, ядро ОС
- Кронос 2 для PDP-11 (QBus)



последовательная линия

Путь через Модулу-0: мимо Бахуса к Кроносу

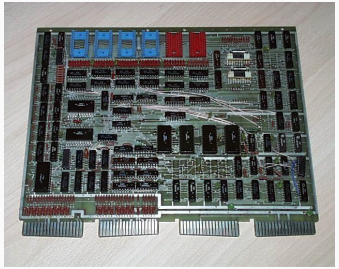


Шаг 1: Электроника-60 (RSX-11)

- КАС (Кронос ассемблер) на Паскале
- Модула-0 компилятор без семантического анализа на КАСе: 3 разработчика/2 недели (пиши правильно!)

Шаг-2: Кронос (мини ОС на КАСе)

- Модула-2 на Модуле-0 (магическое превращение в Модулу-2)
- Excelsior OS на Модуле-2 => Excelsior iV



Модула-0: Уроки этой басни

Внесение (правильных) ограничений может сократить трудоемкость разработки в десятки и сотни раз.

Использовано:

- Везде, но особенно:
- Вир/a0
- Тривиль

Модула-2 => Модула-X: Динамические массивы (ОС и библиотеки)

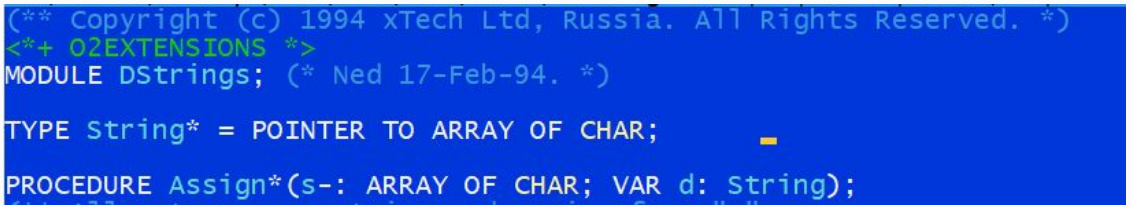
```
// Модула-2
PROCEDURE p1(a: ARRAY OF INTEGER);
BEGIN
END p1;

VAR a: ARRAY [0..7] OF INTEGER; (* константы *)
```

- Как выделять буфера в ОС?
- Как делать динамические массивы?
- Как делать хеш-таблицы?

```
// Модула-X
VAR a: POINTER TO ARRAY OF INTEGER;

NEW(a, 15);
```



Модула-2: Ужасный вывод

C	`printf("The factorial of %d is %d\n", i, fact(i));`
Go	`fmt.Printf("The factorial of %d is %d\n", i, fact(i))`
Typescript	`console.log(`The factorial of ${i} is ${fact(i)}`)`

Модула-2	`InOut.WriteString ("The factorial of ");`
	`InOut.WriteCard (i, 2);`
	`InOut.WriteString (" is ");`
	`InOut.WriteCard (fact(i), 0);`
	`InOut.WriteLn;`

Модула-X: Удобный вывод и строковые операции

C	`printf("The factorial of %d is %d\n", i, fact(i));`
Go	`fmt.Printf("The factorial of %d is %d\n", i, fact(i))`
Typescript	`console.log(`The factorial of ${i} is ${fact(i)}`)`
Модула-2	`ужас-ужас`
Модула-X	`Printf.printf("The factorial of %2d is %d\n", i, r);`

```
(* Copyright (c) 1996 xTech Ltd, Russia. All Rights Reserved. *)
<* +M2EXTENSIONS *>
DEFINITION MODULE Printf;

IMPORT SYSTEM, IOChan;

TYPE ChanId = IOChan.ChanId;

PROCEDURE fprintf(file: ChanId; format: ARRAY OF CHAR; SEQ args: SYSTEM.BYTE);

PROCEDURE printf(format: ARRAY OF CHAR; SEQ args: SYSTEM.BYTE);

PROCEDURE sprintf(VAR buf: ARRAY OF CHAR; format: ARRAY OF CHAR; SEQ args: SYSTEM.BYTE);

END Printf.
```

Модула-X: Уроки этой басни

- Точечные и органичные расширения языка могут существенно увеличить выразительность языка и продуктивность разработчика.
- Есть конструкции, которые обязательно должны быть.

Использовано:

- Везде, но особенно:
- Первый корпоративный
- Тривиль

Вир/а0: Сборка на максимум - проблемы

2003-2006:

- Эксперименты со сборочным программированием
- Язык для разработки компонент: Delphi
- Бинарные компоненты: DLL
- Схема: XML-like

Итог:

- ужас-ужас: винда и дельфи сопротивляются
- странные ошибки, непредсказуемое поведения, зависимости
- большой объем каждой компоненты - в каждой DLL куча библиотек
- очень сложно, не технологично

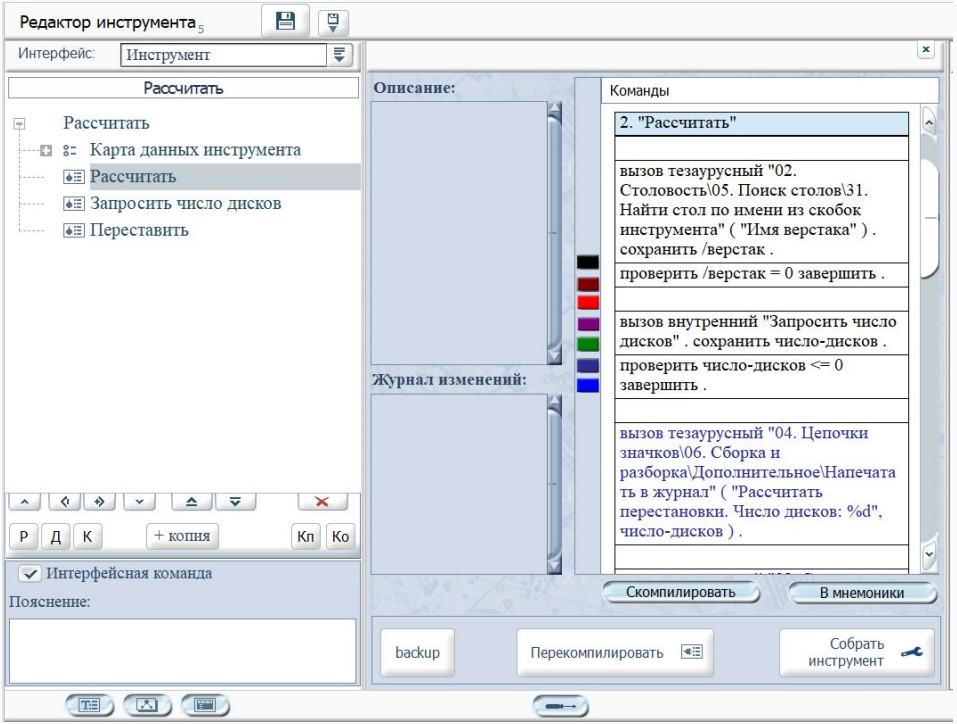
Вир/а0: Сборка на максимум - решение

Решение:

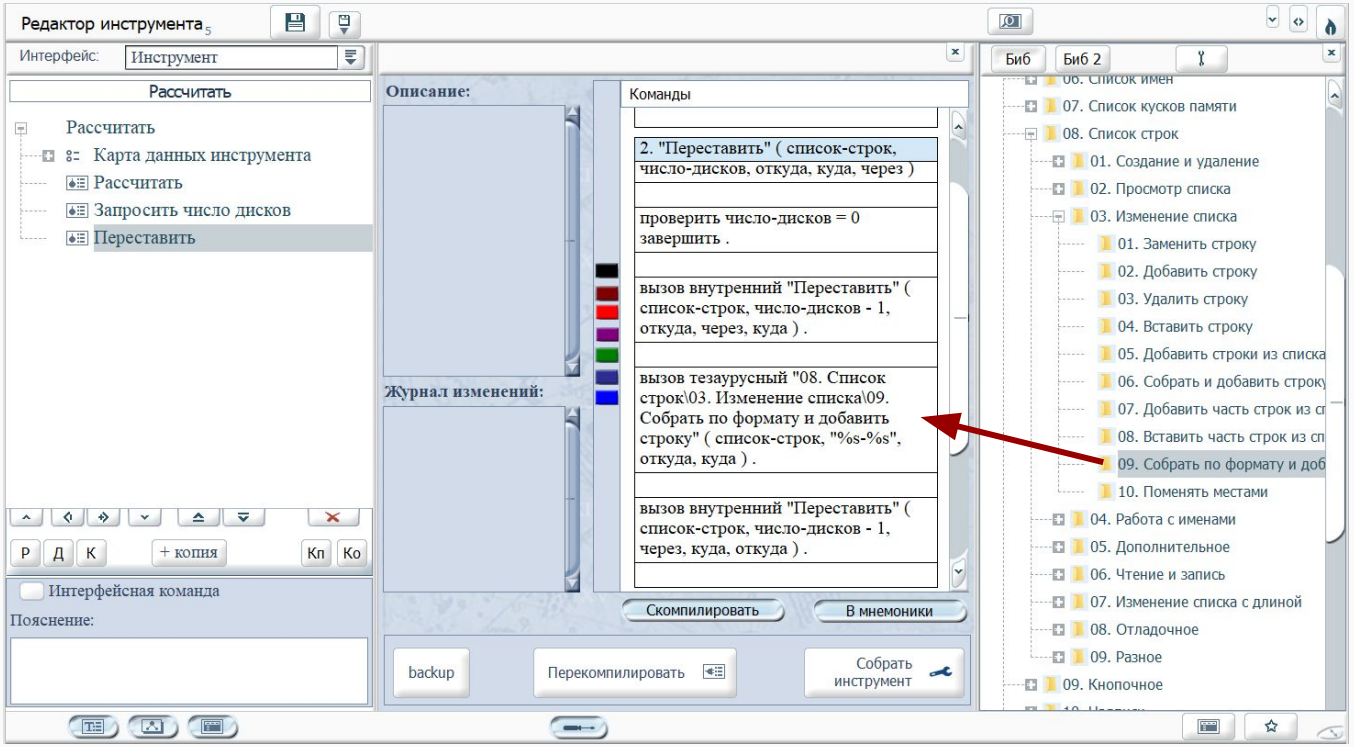
- Выбросить все лишнее, как минимум: Delphi и DLL
- Простейший язык программирования
- Максимально простой компилятор (чтобы не отвлекаться на дополнительные работы):
  - Табличный разбор
  - Генерация для простой стековой машины
  - Табличная эмуляция команд стековой машины через команды x86
- Простейший бинарный формат, в котором есть только необходимое
- Слой, отделяющий от платформы (Windows)

Вир/а0: Язык

- Русский язык с пробелами (ЯРМО)
- Разделение семантики по синтаксису:
  - вызов внутренний
  - вызов тезаурусный
  - вызов …
- Типовая система: отсутствует! (FORTH)
  - Один тип данных: Слово64
  - Целое число или указатель
  - Вещественные через вызовы
- Нет описаний переменных
- Семантический анализ: отсутствует! (как в Модуле-0 - пиши правильно)
- Правое присваивание (легче генерить)
- Оператор “проверить” (guard, гл. шампур)
- Структура привязана к среде разработки



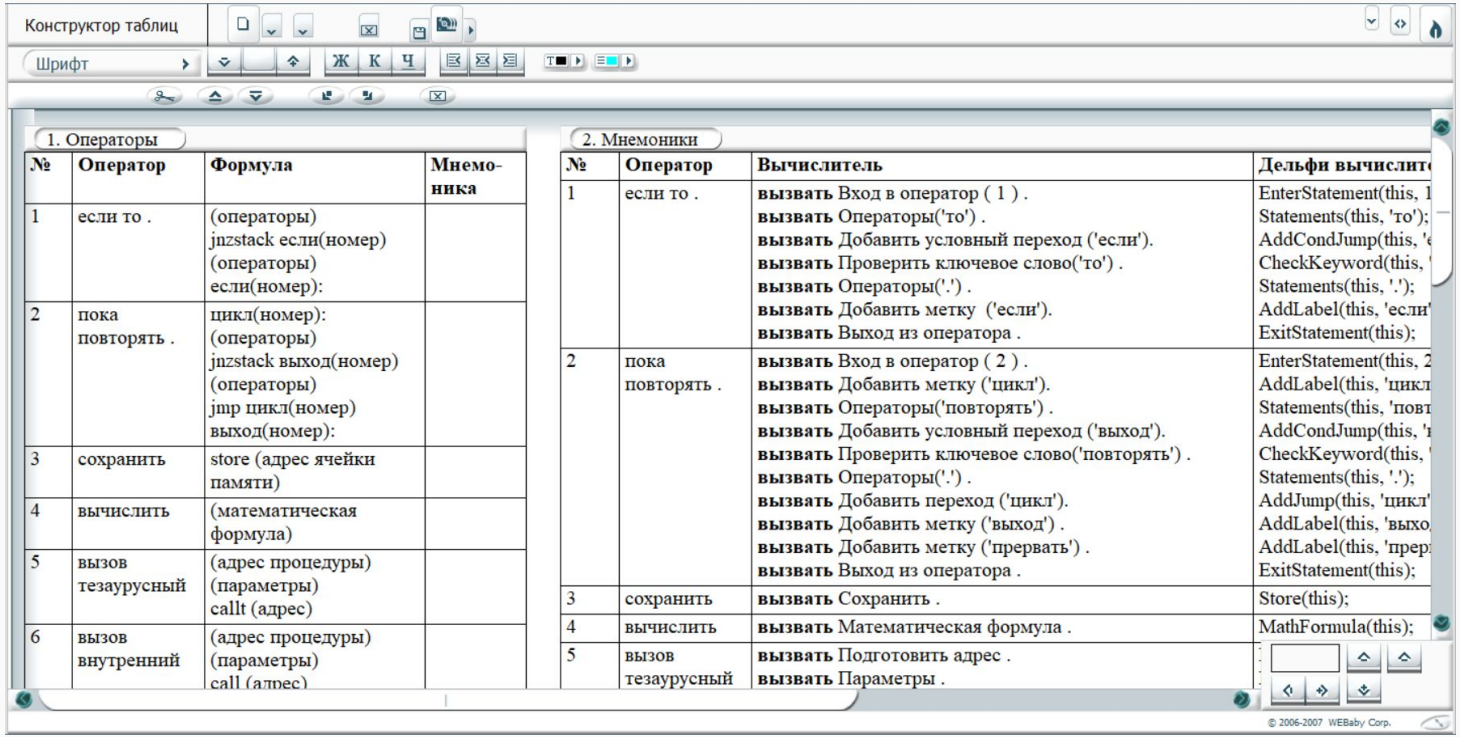
Вир/а0: Язык. Ничего встроенного. Сборка!



Сборка на всех уровнях:

- Инструмент
- Тезаурус
- GUI
- Сервис
  - Отладка
  - Обновление
  - …
- Программа

Вир/а0: Табличный компилятор и табличный ассемблер



1. Операторы

№	Оператор	Формула	Мнемо-ника
1	если то .	(операторы) jnzstack если(номер) (операторы) если(номер):	
2	пока повторять .	цикл(номер): (операторы) jnzstack выход(номер) (операторы) jmp цикл(номер) выход(номер):	
3	сохранить	store (адрес ячейки памяти)	
4	вычислить	(математическая формула)	
5	вызов тезаурусный	(адрес процедуры) (параметры) callt (адрес)	
6	вызов внутренний	(адрес процедуры) (параметры) call (адрес)	

2. Мнемоники

№	Оператор	Вычислитель	Дельфи вычислит
1	если то .	**вызвать** Вход в оператор (1) . **вызвать** Операторы('то') . **вызвать** Добавить условный переход ('если'). **вызвать** Проверить ключевое слово('то') . **вызвать** Операторы('.') . **вызвать** Добавить метку ('если'). **вызвать** Выход из оператора .	EnterStatement(this, 1 Statements(this, 'то'); AddCondJump(this, 'е CheckKeyword(this, ' Statements(this, '.'); AddLabel(this, 'если' ExitStatement(this);
2	пока повторять .	**вызвать** Вход в оператор (2) . **вызвать** Добавить метку ('цикл'). **вызвать** Операторы('повторять') . **вызвать** Добавить условный переход ('выход'). **вызвать** Проверить ключевое слово('повторять') . **вызвать** Операторы('.') . **вызвать** Добавить переход ('цикл'). **вызвать** Добавить метку ('выход') . **вызвать** Добавить метку ('прервать') . **вызвать** Выход из оператора .	EnterStatement(this, 2 AddLabel(this, 'цикл Statements(this, 'повт AddCondJump(this, 'в CheckKeyword(this, ' Statements(this, '.'); AddJump(this, 'цикл' AddLabel(this, 'выхо AddLabel(this, 'прер ExitStatement(this);
3	сохранить	**вызвать** Сохранить .	Store(this);
4	вычислить	**вызвать** Математическая формула .	MathFormula(this);
5	вызов тезаурусный	**вызвать** Подготовить адрес . **вызвать** Параметры .	

- ”если” без “иначе”
- один цикл: “пока”
- “сохранить” для “:=”
- разные вызовы
- явное начало выражения
- вызов - не часть выражения

явное начало выражения: x = 1 + 2 => вычислить 1 + 2 . сохранить x .

Вир/а0: Уроки этой басни

- Ограничения языка и компилятора делают чудеса:
  - Язык и компилятор сделаны за 1 неделю
  - Язык впоследствии расширялся, подход и устройство компилятора не менялось
- Убирание зависимостей
  - Очень сильно влияет на скорость и качество разработки
  - Этот урок уже был дан в Кроносе, но был осознан только здесь
- Изменение условий задачи (сборка - “странная” задача)
  - требует пересмотра привычных решений (нужны “странные” решения)

Использовано:

- Минимизация зависимостей: везде, Тривиль
- Поиск непривычных решений: везде, Тривиль, Арвиль

Вопросы

EOF