



ИСПОЛЬЗОВАНИЕ БЕЗОПАСНОЙ СИСТЕМЫ СБОРКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

О СЕБЕ



Хмелёв Игорь Геннадьевич

- Инженер
- Более 10 лет в IT и кибербезопасности
- Занимал различные позиции при разработке программного обеспечения
- Активно принимаю участие в CTF, конференциях и профильных мероприятиях
- Team Lead и архитектор в продукте класса NGFW

ВОДИЧКА (ИСТОРИЯ ПО)

Программа | Алгоритм

– заданная последовательность действий

Программное обеспечение (ПО) – набор инструкций для выполнения действий на вычислительной машине

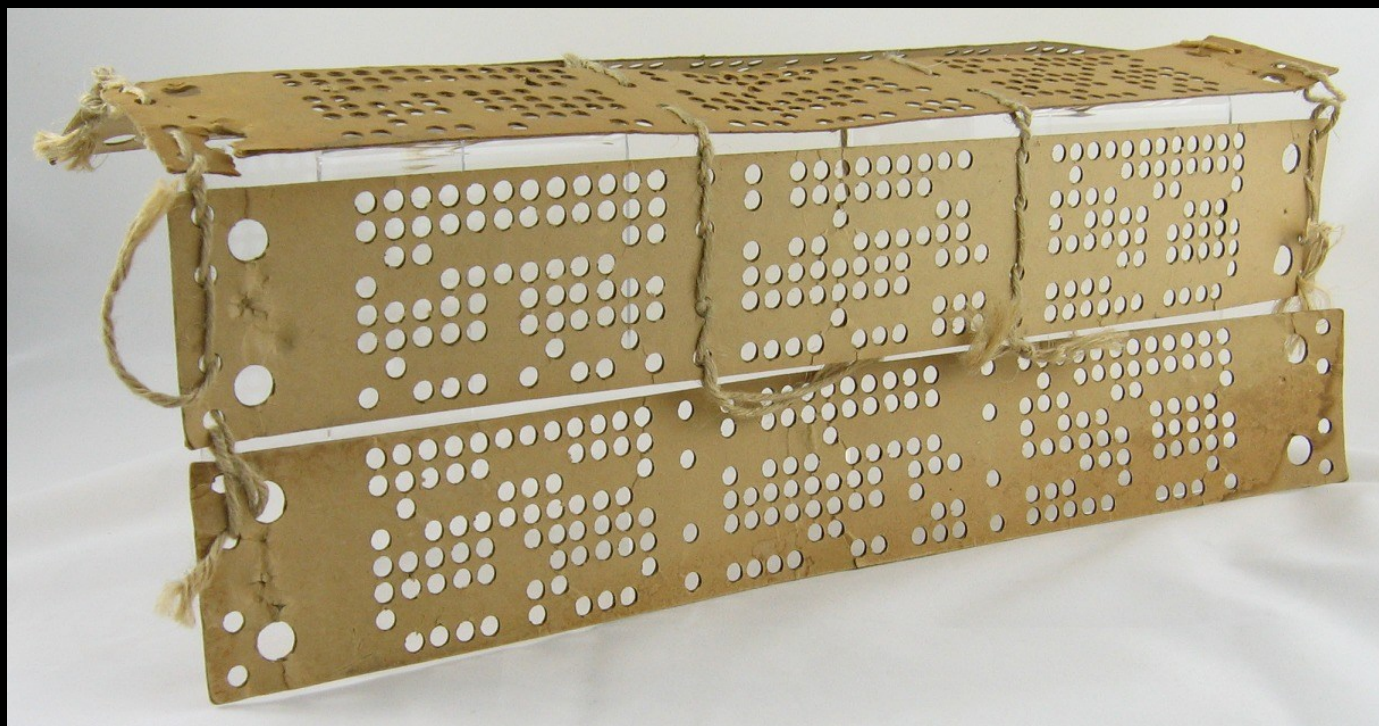
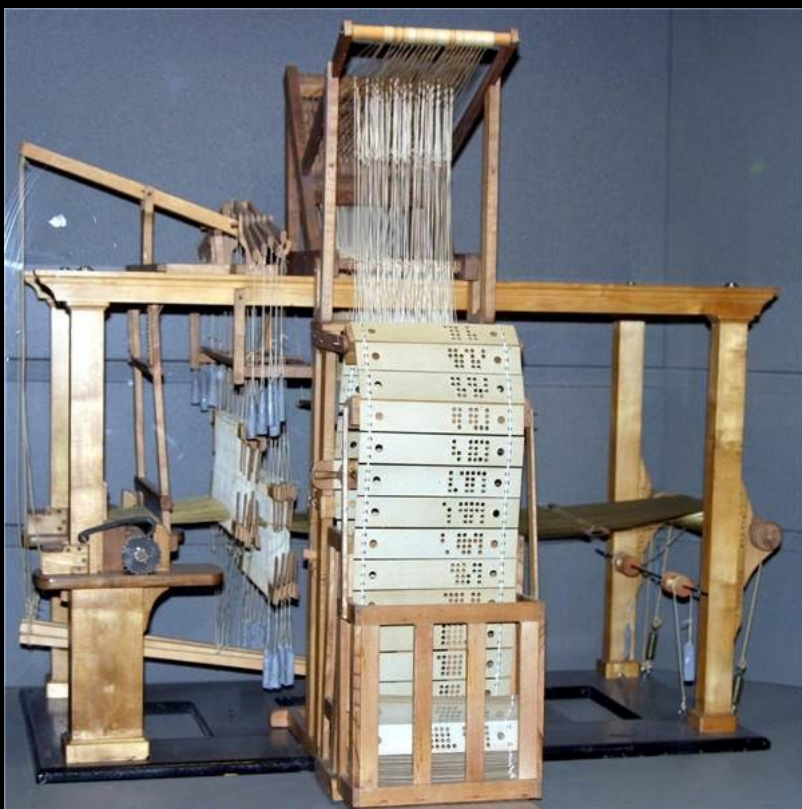
ПЕРВЫЕ ПРОГРАММЫ

- Инструкции для музыкальных инструментов



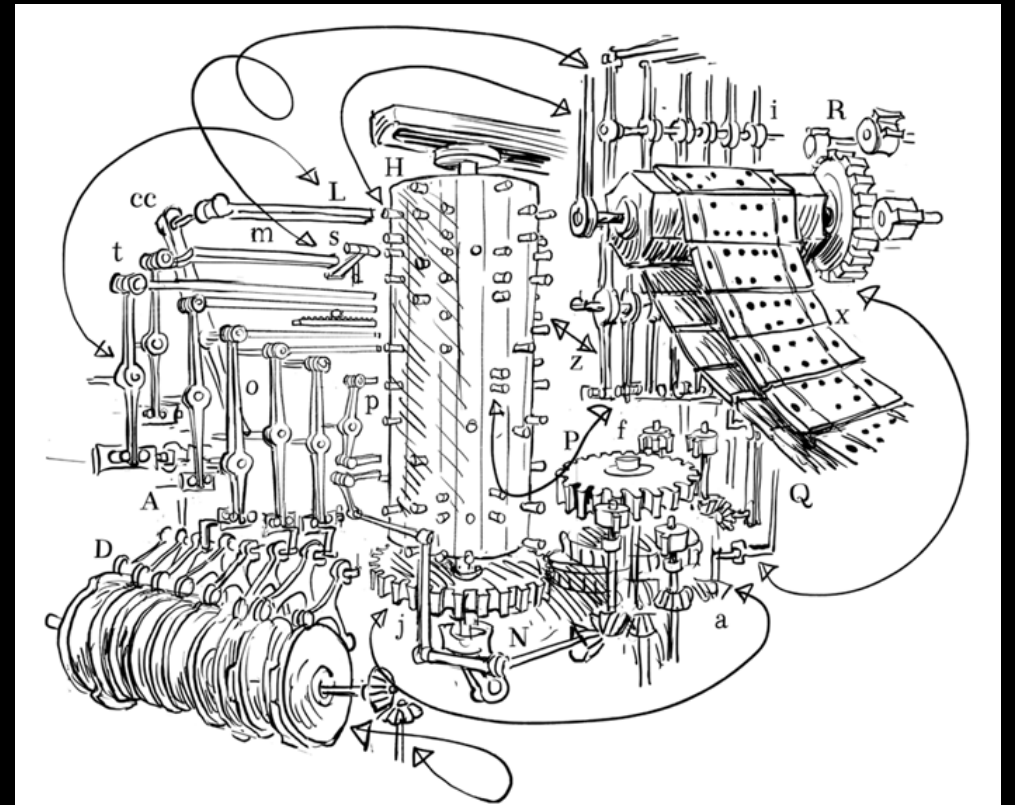
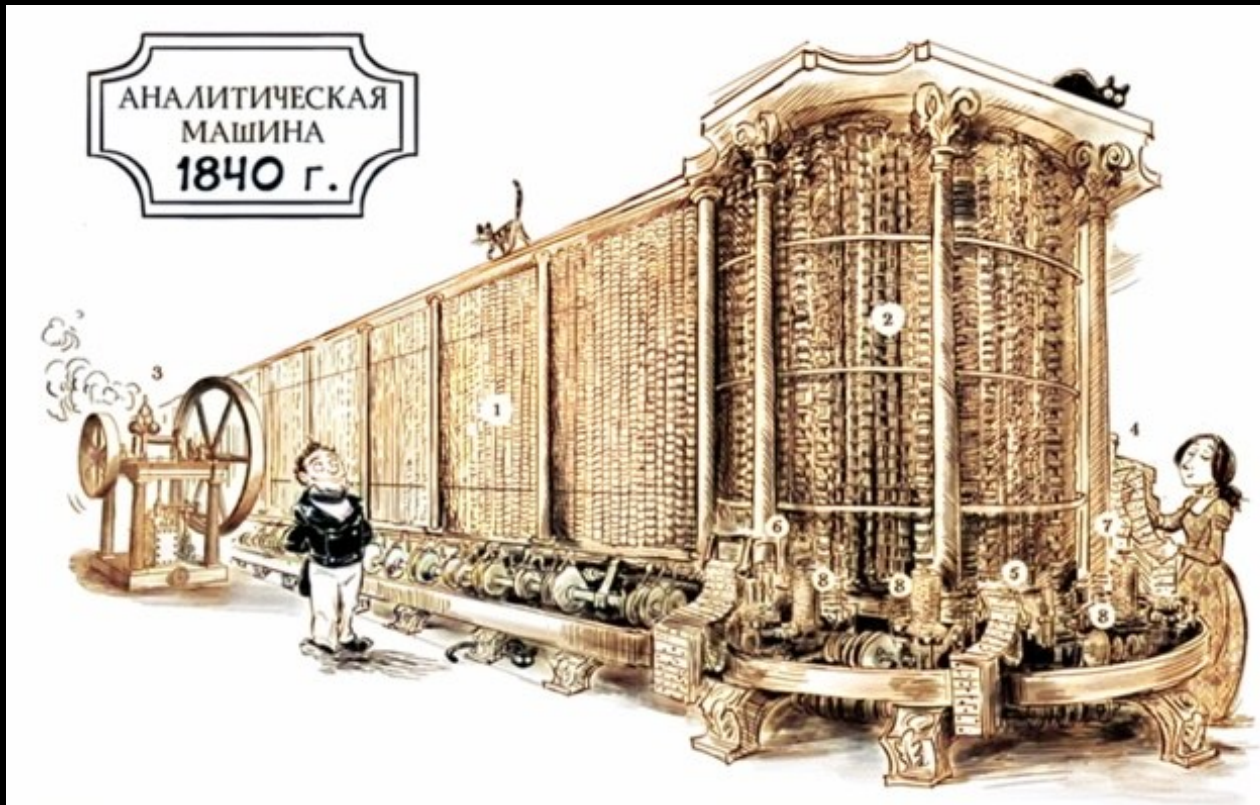
ПЕРВЫЕ ПРОГРАММЫ

- 1801 год – инструкция для станков (Жаккардовый ткацкий станок)



ПЕРВЫЕ ПРОГРАММЫ

- 1830е-1840е – Аналитическая машина Бэббиджа



ПЕРВЫЕ ПРОГРАММЫ

- 1830е-1840е – первая программа (Ада Лавлейс)

ADA LOVELACE FIRST COMPUTER PROGRAMMER



The Analytical Engine

Lovelace's program turned a complex formula into simple calculations that could be encoded on punched cards and fed into Charles Babbage's Analytical Engine, a mechanical computer that he designed but never built. She published it in 1843, a century before the modern computer age.

"I want to put in something about Bernoulli's Number, in one of my Notes, as an example of how an explicit function may be worked out by the engine, without having been worked out by human head and hands first."



A Universal Computer

Lovelace did more than write the first computer program. She was also the first person to realise that a general purpose computer could do anything, given the right data and instructions.

"The Analytical Engine weaves algebraic patterns just as the Jacquard loom weaves flowers and leaves."

"Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent."



Augusta Ada King,
Countess of Lovelace
Born: 10 December 1815
Died: 27 November 1852

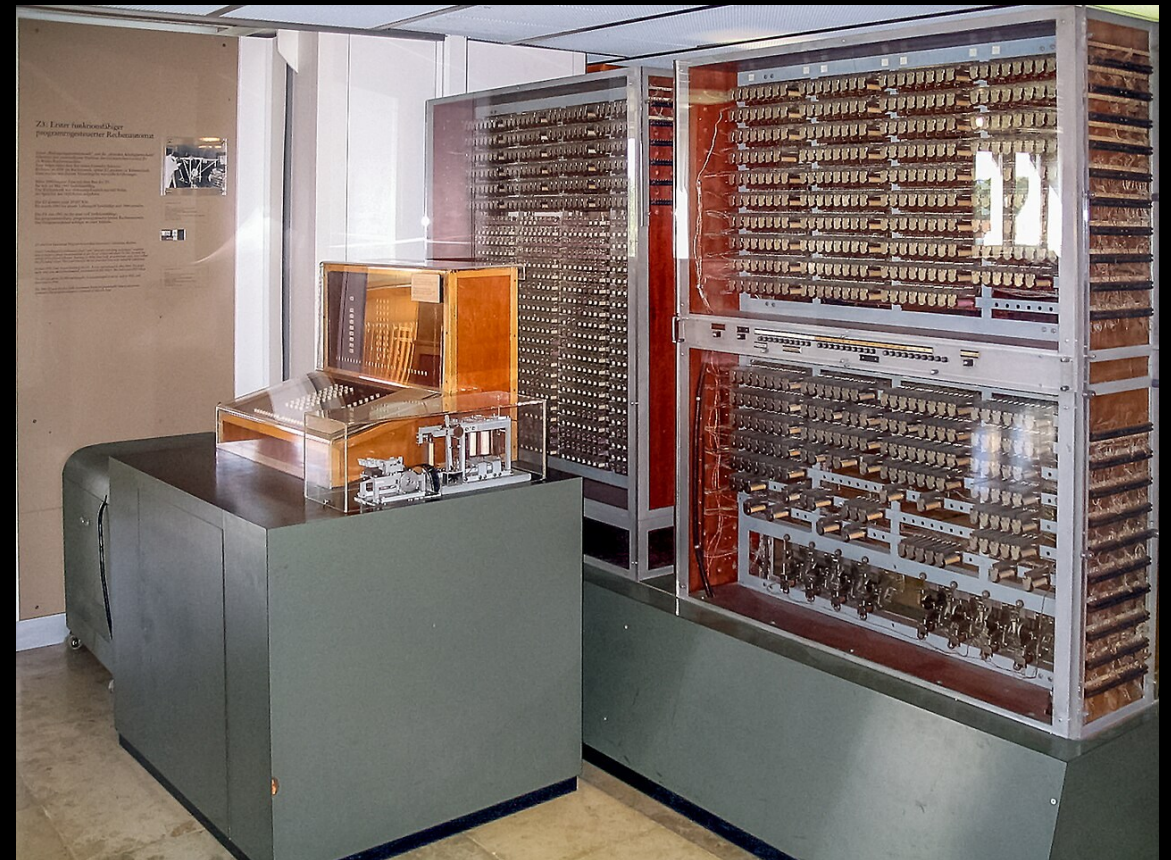
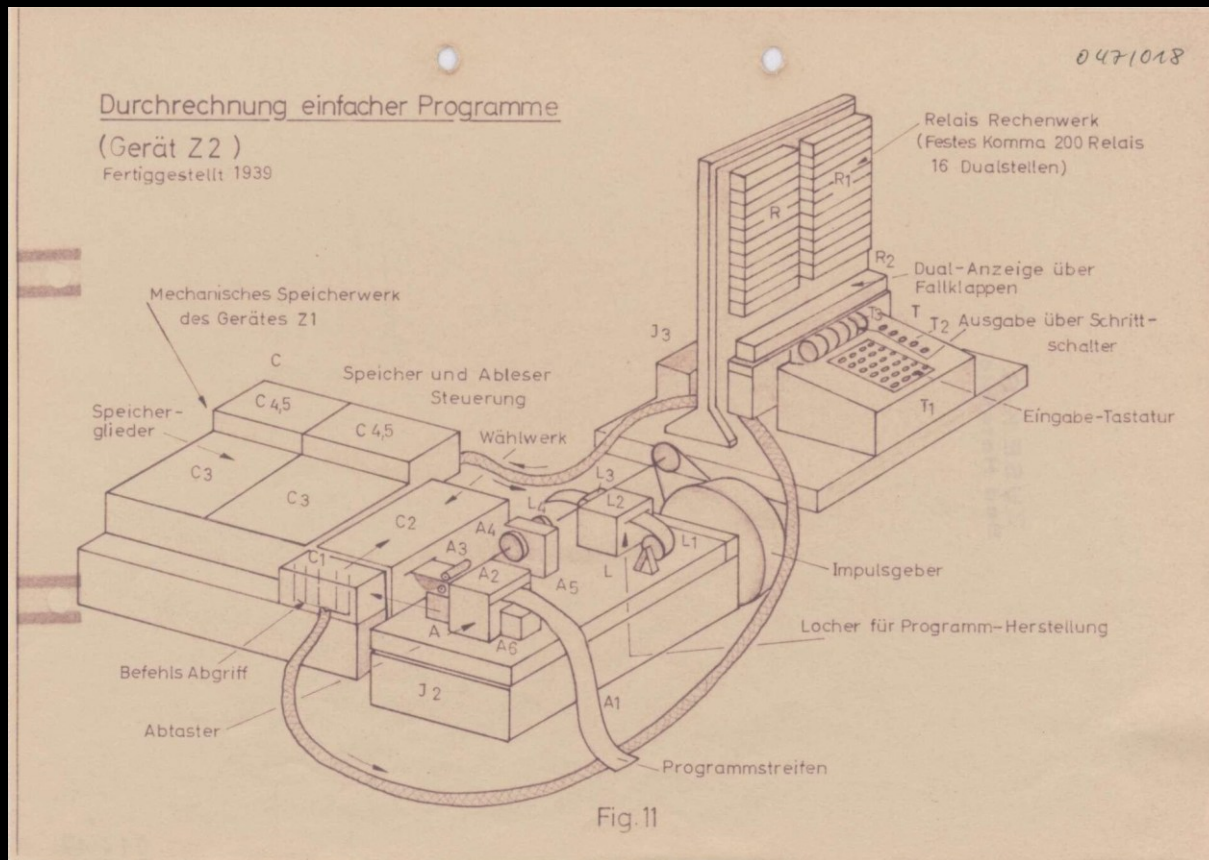


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 et seq.)

Number of Operation.	Nature of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.	Data.										Working Variables.										Result Variables.			
						$1V_1$	$1V_2$	$1V_3$	$1V_4$	$1V_5$	$1V_6$	$1V_7$	$1V_8$	$1V_9$	$1V_{10}$	$1V_{11}$	$1V_{12}$	$1V_{13}$	$1V_{14}$	$1V_{15}$	$1V_{16}$	$1V_{17}$	$1V_{18}$	$1V_{19}$	$1V_{20}$	$1V_{21}$	$1V_{22}$	$1V_{23}$	$1V_{24}$
						1	2	n																					
1	$\times V_2 \times V_3$	$1V_4 = 1V_2 \times 1V_3$	$1V_4 = 1V_2 \times 1V_3$		$-2n$...	2	n	2n	2n	2n																		
2	$-V_4 - V_5$	$2V_4 = 1V_4 - 1V_5$	$2V_4 = 1V_4 - 1V_5$		$-2n-1$...	1																		
3	$+V_4 + V_5$	$3V_4 = 2V_4 + 1V_5$	$3V_4 = 2V_4 + 1V_5$		$2n+1$...	1																		
4	$+V_4 + V_4$	$4V_4 = 3V_4 + 1V_4$	$4V_4 = 3V_4 + 1V_4$		$2n+1$																		
5	$+V_4 + V_4$	$5V_4 = 4V_4 + 1V_4$	$5V_4 = 4V_4 + 1V_4$		$2n+1$...	2																		
6	$-V_{13} - V_{11}$	$6V_{13} = 5V_{13} - 1V_{11}$	$6V_{13} = 5V_{13} - 1V_{11}$		$-1 \cdot \frac{2n-1}{2} \cdot \frac{2n-1}{2} = A_0$																		
7	$-V_{13} - V_4$	$7V_{13} = 6V_{13} - 1V_4$	$7V_{13} = 6V_{13} - 1V_4$		$n-1 (=3)$...	1	...	n																		
8	$+V_4 + V_2$	$8V_{13} = 7V_{13} + 1V_2$	$8V_{13} = 7V_{13} + 1V_2$		$-2+0=2$...	2																		
9	$+V_4 + V_2$	$9V_{13} = 8V_{13} + 1V_2$	$9V_{13} = 8V_{13} + 1V_2$		$2n = A_1$																		
10	$\times V_{13} \times V_{11}$	$10V_{13} = 9V_{13} \times 1V_{11}$	$10V_{13} = 9V_{13} \times 1V_{11}$		$B_1 \cdot \frac{2n}{2} = B_1 A_1$																		
11	$+V_{13} + V_{13}$	$11V_{13} = 10V_{13} + 1V_{13}$	$11V_{13} = 10V_{13} + 1V_{13}$		$-1 \cdot \frac{2n-1}{2} + B_1 \cdot \frac{2n}{2}$																		
12	$-V_{13} - V_4$	$12V_{13} = 11V_{13} - 1V_4$	$12V_{13} = 11V_{13} - 1V_4$		$n-2 (=2)$...	1																		
13	$-V_4 - V_4$	$13V_{13} = 12V_{13} - 1V_4$	$13V_{13} = 12V_{13} - 1V_4$		$-2n-1$...	1																		
14	$+V_4 + V_2$	$14V_{13} = 13V_{13} + 1V_2$	$14V_{13} = 13V_{13} + 1V_2$		$-2+1=3$...	1																		
15	$+V_4 + V_2$	$15V_{13} = 14V_{13} + 1V_2$	$15V_{13} = 14V_{13} + 1V_2$		$2n-1$...	1																		
16	$\times V_4 \times V_{11}$	$16V_{13} = 15V_{13} \times 1V_{11}$	$16V_{13} = 15V_{13} \times 1V_{11}$		$\frac{2n-1}{2} \cdot \frac{2n-1}{2}$																		
17	$-V_4 - V_4$	$17V_{13} = 16V_{13} - 1V_4$	$17V_{13} = 16V_{13} - 1V_4$		$2n-2$...	1																		
18	$+V_4 + V_2$	$18V_{13} = 17V_{13} + 1V_2$	$18V_{13} = 17V_{13} + 1V_2$		$-3+1=4$...	1																		
19	$+V_4 + V_2$	$19V_{13} = 18V_{13} + 1V_2$	$19V_{13} = 18V_{13} + 1V_2$		$2n-2$																		
20	$\times V_4 \times V_{11}$	$20V_{13} = 19V_{13} \times 1V_{11}$	$20V_{13} = 19V_{13} \times 1V_{11}$		$\frac{2n-1}{2} \cdot \frac{2n-1}{2} = A_2$																		
21	$\times V_{13} \times V_{11}$	$21V_{13} = 20V_{13} \times 1V_{11}$	$21V_{13} = 20V_{13} \times 1V_{11}$		$B_2 \cdot \frac{2n-1}{2} \cdot \frac{2n-1}{2} = B_2 A_2$																		
22	$+V_{13} + V_{13}$	$22V_{13} = 21V_{13} + 1V_{13}$	$22V_{13} = 21V_{13} + 1V_{13}$		$A_0 + B_1 A_1 + B_2 A_2$																		
23	$-V_{13} - V_4$	$23V_{13} = 22V_{13} - 1V_4$	$23V_{13} = 22V_{13} - 1V_4$		$n-3 (=1)$...	1																		
Here follows a repetition of Operations thirteen to twenty-three.																													
24	$+V_{13} + V_2$	$24V_{13} = 23V_{13} + 1V_2$	$24V_{13} = 23V_{13} + 1V_2$		B_2																		
25	$+V_4 + V_2$	$25V_{13} = 24V_{13} + 1V_2$	$25V_{13} = 24V_{13} + 1V_2$		$n+1 = 4+1=5$...	1	...	n+1																		

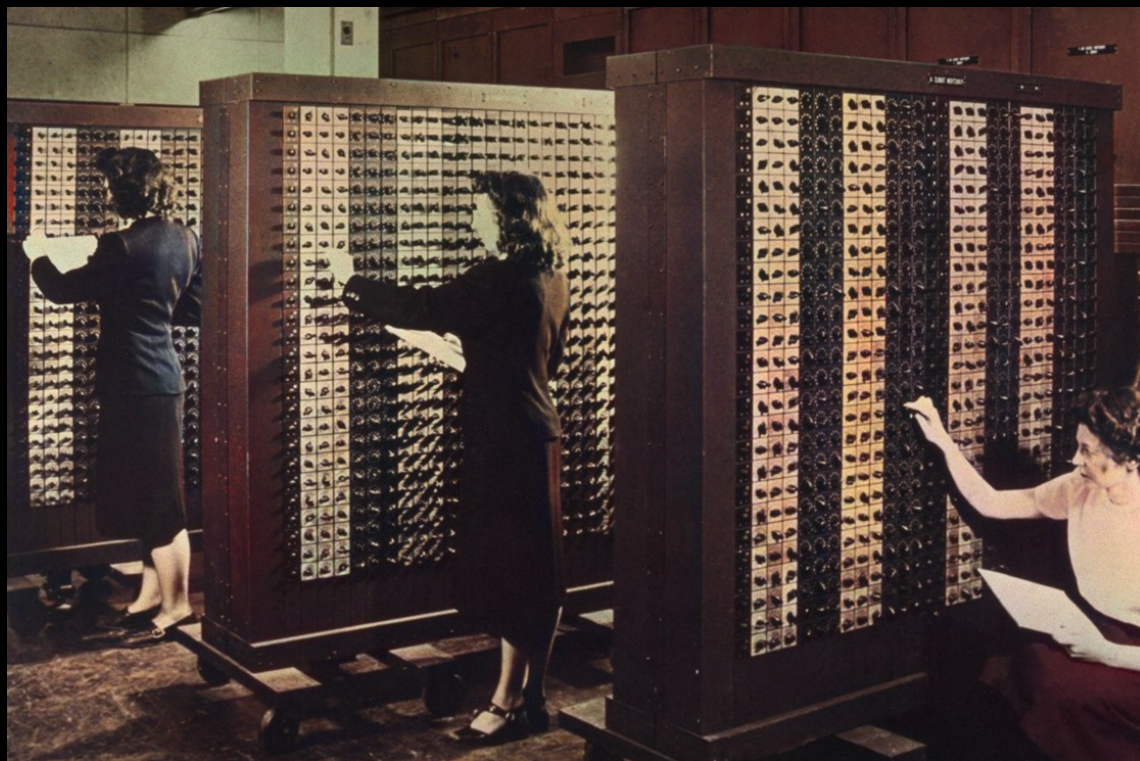
ПЕРВЫЕ ПРОГРАММЫ

- 1940е – электромеханический цифровой компьютер Z2, Z3, Z4



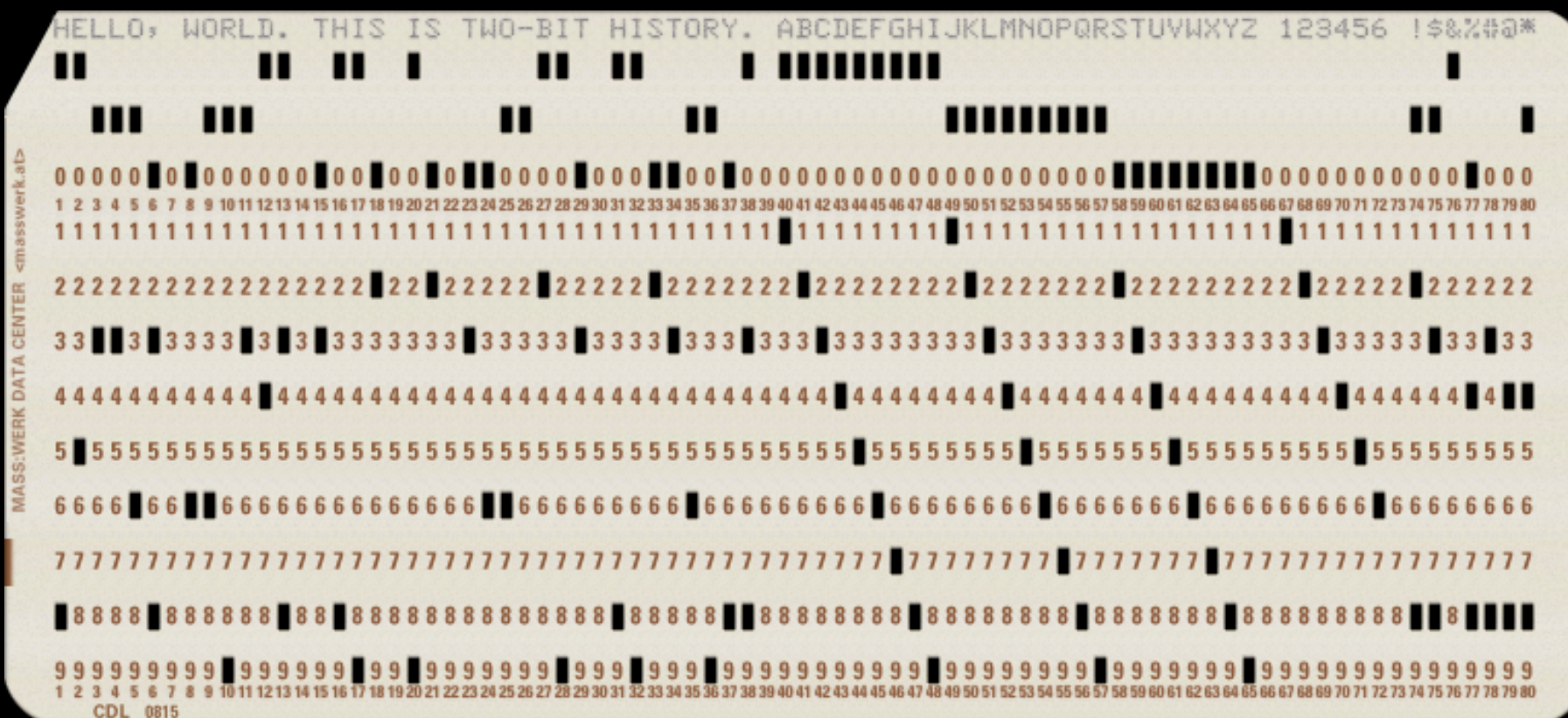
ПЕРВЫЕ ПРОГРАММЫ

- 1945 – ENIAC – электронный числовой интегратор и вычислитель (первый полностью электронный универсальный компьютер)



ПЕРВЫЕ ПРОГРАММЫ

- 1940е – Перфокарты с двоичным кодом



ПЕРВЫЕ ПРОГРАММЫ

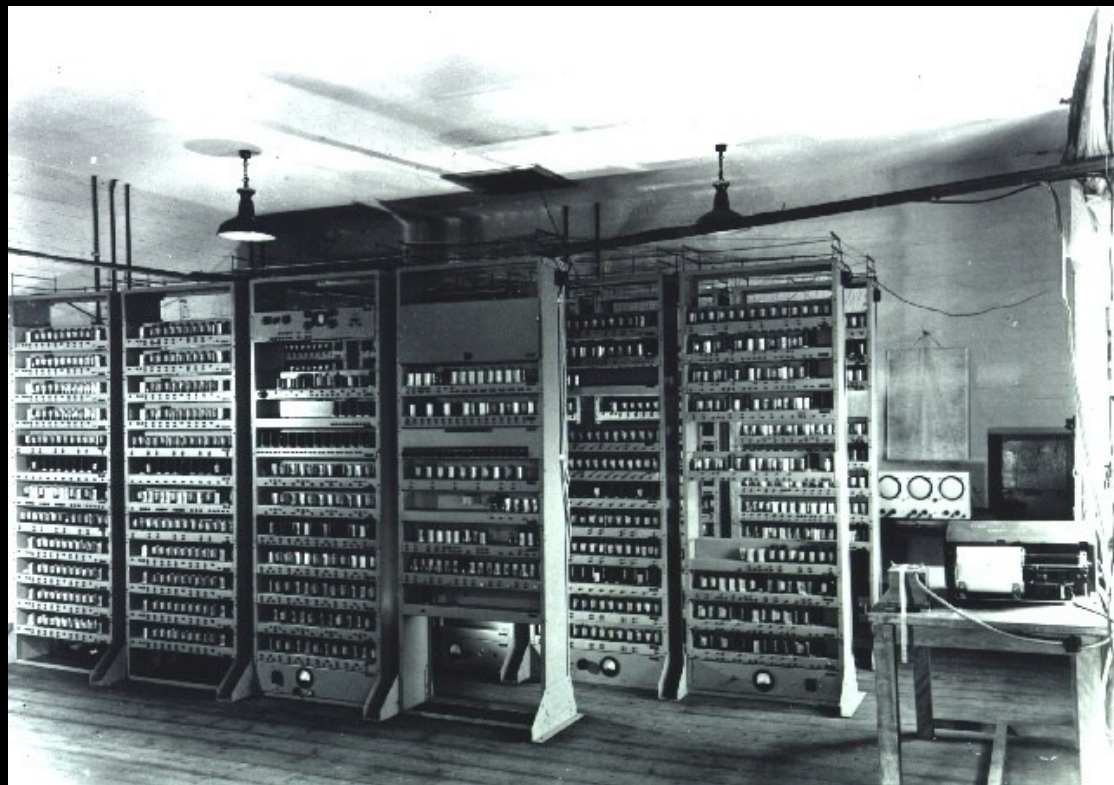
- 1948 – Первый в мире высокоуровневый ЯП – Планкалкюль (для Z4)

Plankalkül

```
P1 max3 (V0[:8.0],V1[:8.0],V2[:8.0]) → R0[:8.0]
max(V0[:8.0],V1[:8.0]) → Z1[:8.0]
max(Z1[:8.0],V2[:8.0]) → R0[:8.0]
END
P2 max (V0[:8.0],V1[:8.0]) → R0[:8.0]
V0[:8.0] → Z1[:8.0]
(Z1[:8.0] < V1[:8.0]) → V1[:8.0] → Z1[:8.0]
Z1[:8.0] → R0[:8.0]
END
```

ПЕРВЫЕ ПРОГРАММЫ

- 1947 – Написание программ на Ассемблер
EDSAC – один из первых компьютер с хранимой в памяти программой



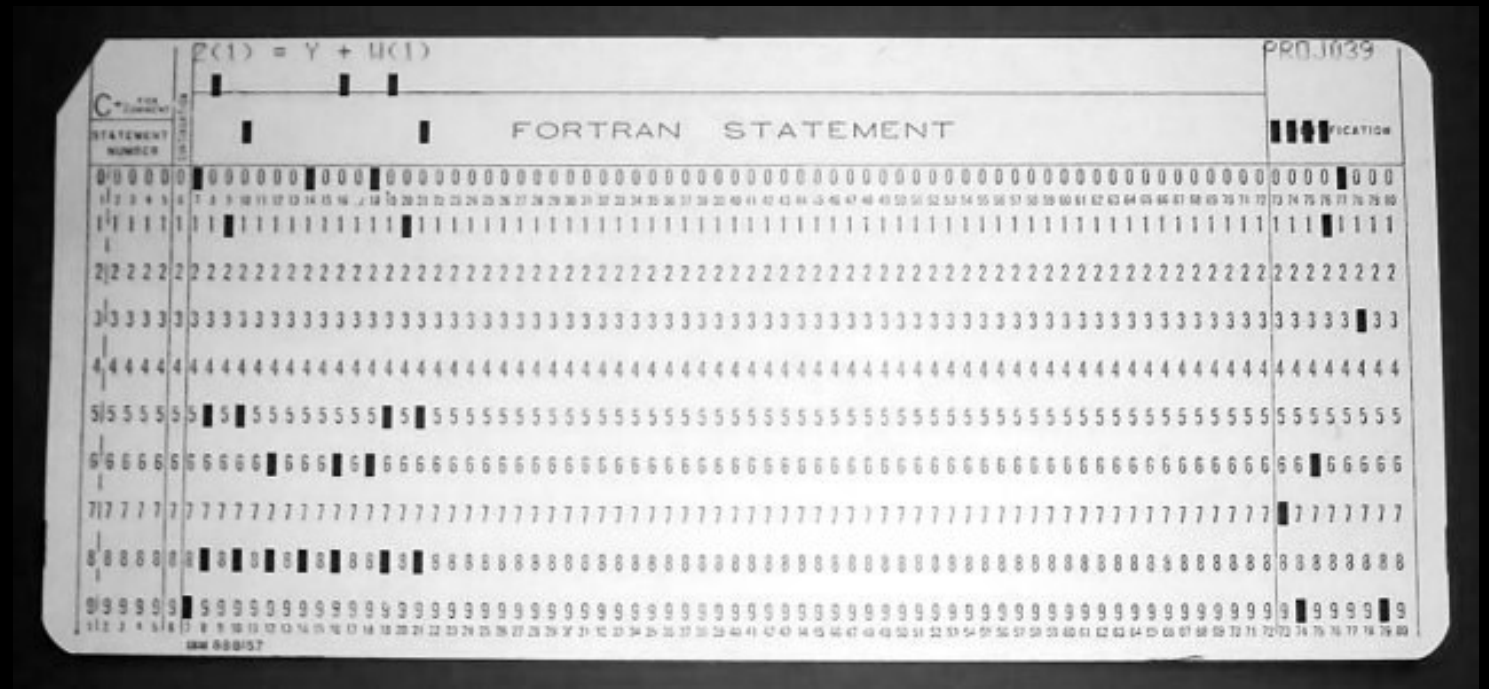
[32] A 69 S [загрузка в аккумулятор степени многочлена]
[33] T 1 S [запись степени многочлена в ячейку 1]
[34] A 71 S [загрузка старшего коэффициента в аккумулятор]
[35] T 2 S [запись старшего коэффициента в рабочую ячейку 2]
[36] H 68 S [запись 1 в умножающий регистр]
[37] V 2 S [умножение числа из ячейки 2 на число в умножающем регистре]
[38] R 1 S [сдвиг числа вправо на 2 бита, для коррекции умножения]
[39] T 2 L [запись старшего коэффициента в длинную ячейку]

[40] A 70 S [загрузка в аккумулятор адреса 1-го элемента массива]
[41] L 0 L [сдвиг аккумулятора, для коррекции]
[42] A 54 S [прибавляем код инструкции с нулевым полем адреса]
[43] T 54 S [запись сформированной инструкции, обнуление аккумулятора]

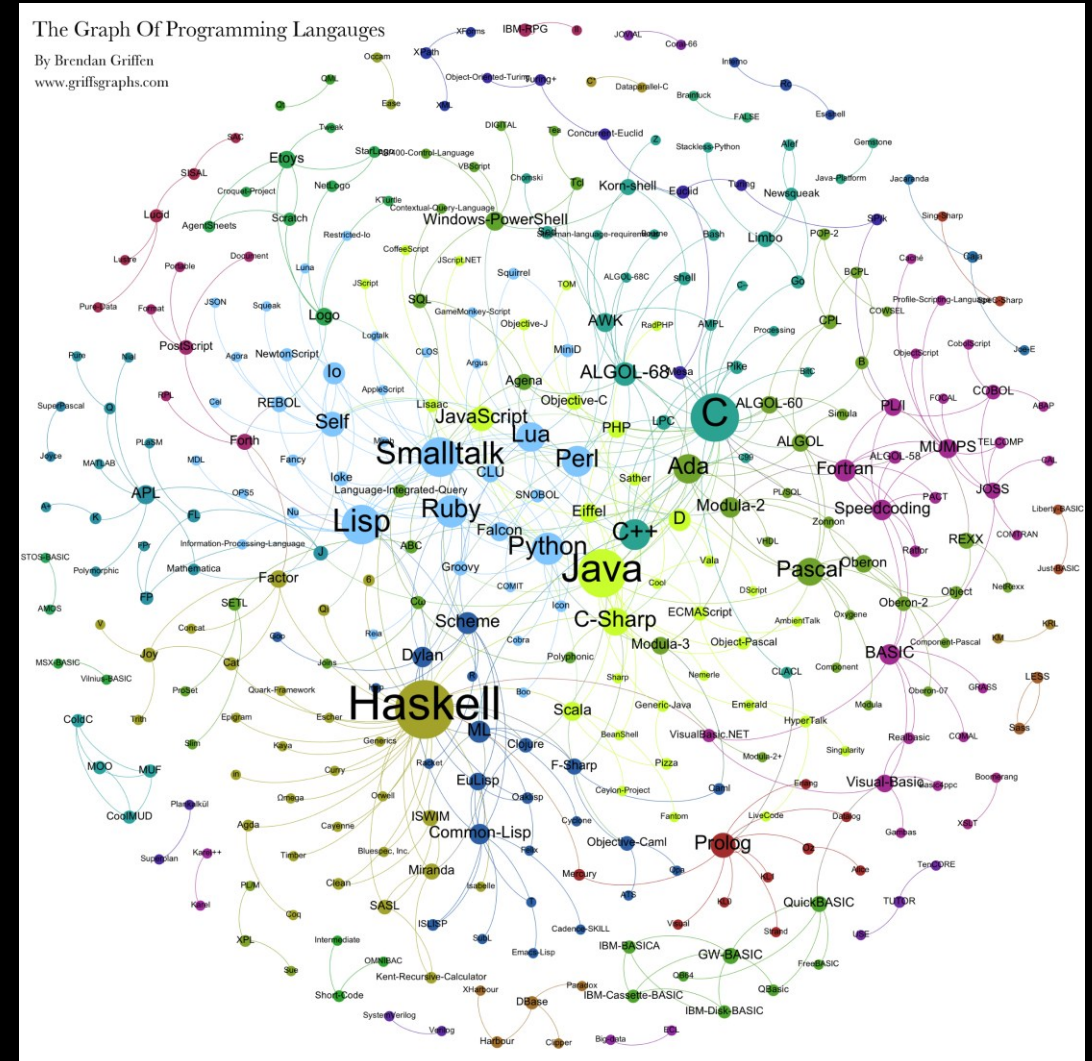
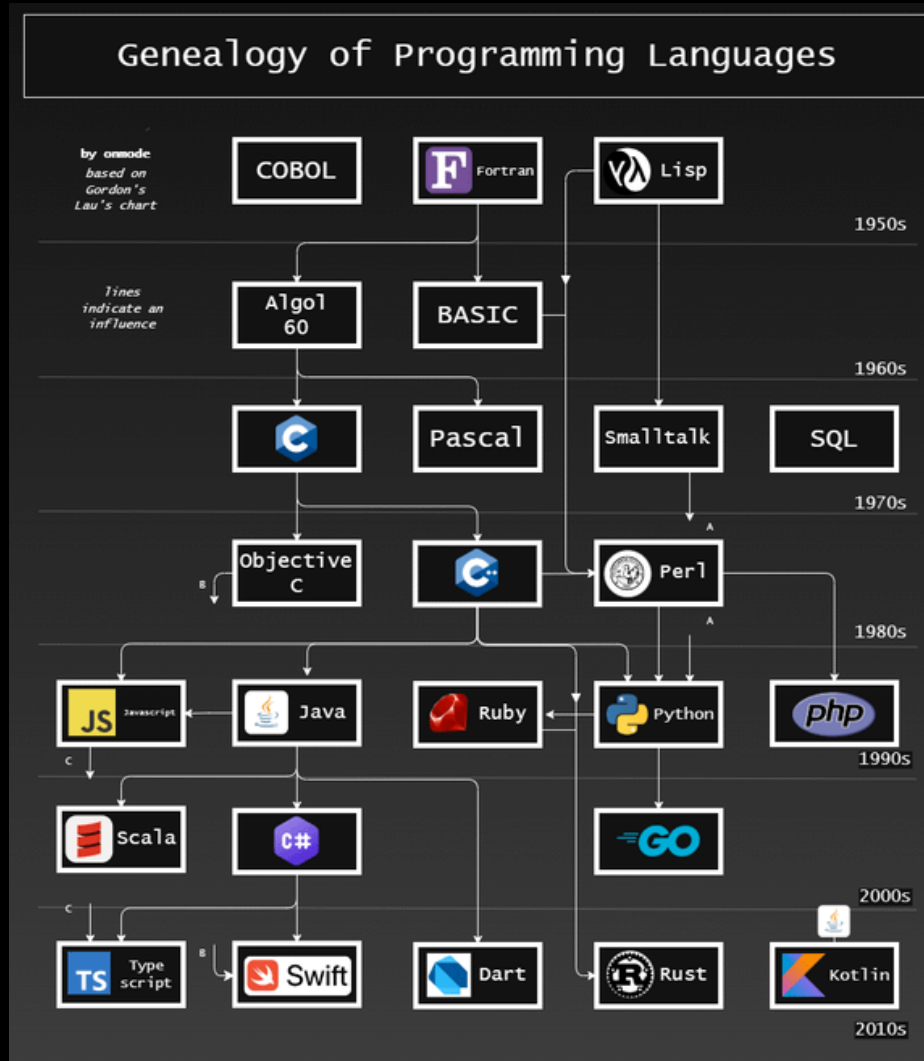
[44 loop] A 1 S [загружаем в аккумулятор счётчик необработанных элементов массива]
[45] S 68 S [вычитание константы = 1]
[46] E 48 S [if (acc >= 0) goto 48 else end]
[47] Z 0 S [точка остановки, завершение программы]

ПЕРВЫЕ ПРОГРАММЫ

- 1957 – Высокоуровневый ЯП FORTRAN + Компилятор



ПЕРВЫЕ ПРОГРАММЫ



ПЕРВЫЕ ПРОГРАММЫ

Функциональные языки программирования (Лямбда-Исчисления)

- Программные средства доказательства теорем
- ЯП с формальной верификацией

Примеры ЯП для формальной верификации:

- Coq (Gallina)
- ML (Meta Language)
- OCaml

Пример использования:

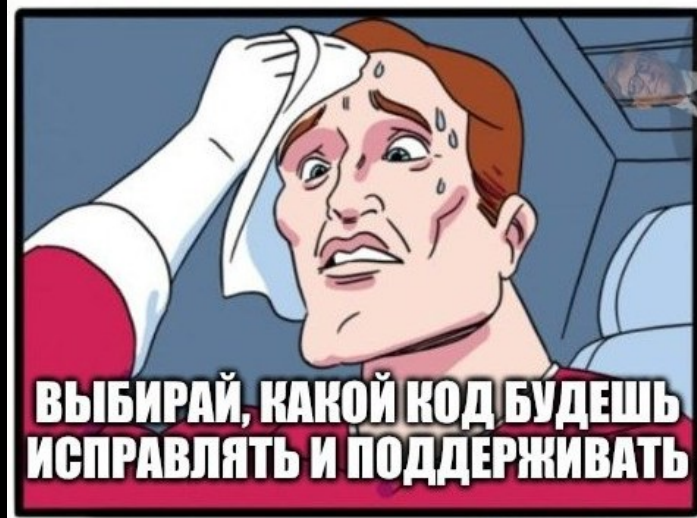
- Формальное доказательство математических задач
- Основной код пишется на одном ЯП, а формальная верификация делается на другом ЯП

ПЕРВЫЕ ПРОГРАММЫ

Альтернативные подходы:

- Low-code – визуальные инструменты + код
- No-code – визуальные инструменты
- Vibe coding – написание программ с помощью ИИ
- Гибридный подход

ПЕРВЫЕ ПРОГРАММЫ





ВИДЫ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

ВИДЫ ЯП

По уровню абстракции:

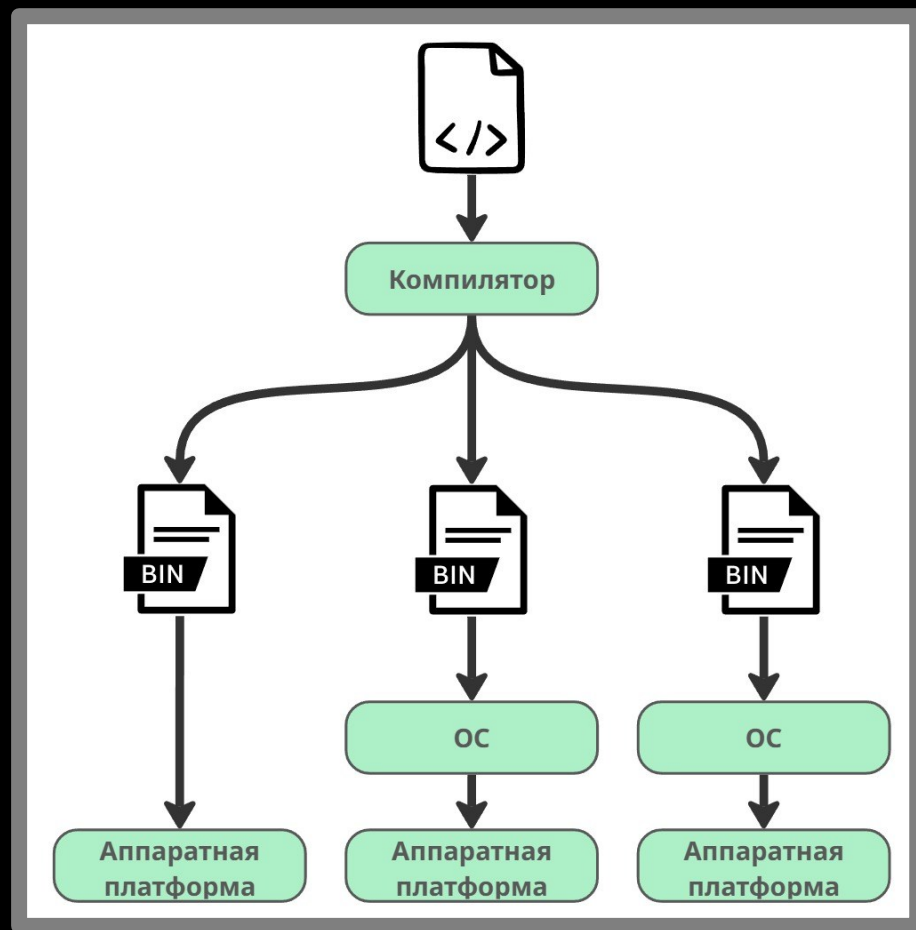
- Машинный код | Машинные инструкции
- Промежуточное представление (ассемблер, байт-код)
- Высокоуровневый ЯП
- Упрощенный интерфейс написания кода (диаграммы, чаты)

По способу реализации:

- Декларативные и императивные
- Строгая и нестрогая типизация (типобезопасность)
- Компилируемые ЯП
- Интерпретируемые ЯП
- Гибридные ЯП
- Транспайлер

ВИДЫ ЯП

- Компилируемые ЯП



Примеры: C, C++, Rust, Go, Fortran

Плюсы:

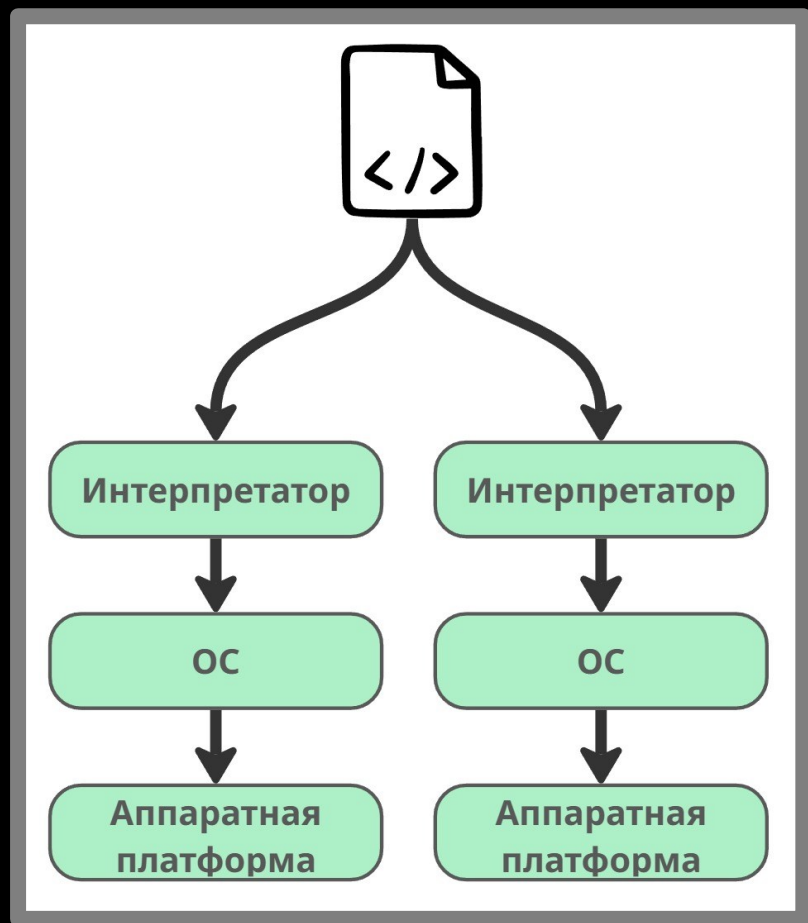
- Производительность
- Оптимизация
- Защита исходного кода

Минусы:

- Зависимость от платформы
- Сложность отладки

ВИДЫ ЯП

- Интерпретируемые ЯП



Примеры: BASH, JavaScript, PHP

Плюсы:

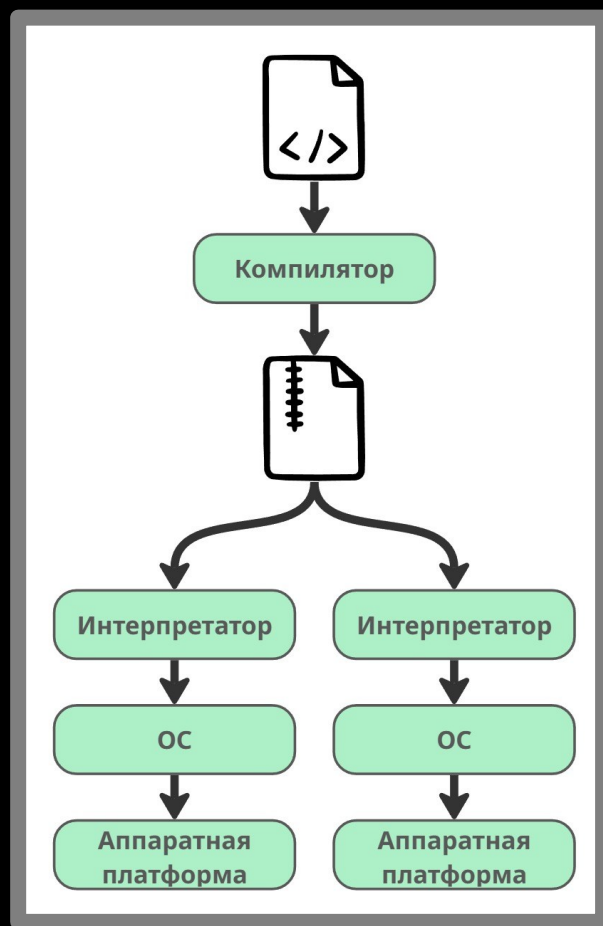
- Кроссплатформенность
- Простота отладки

Минусы:

- Меньшая производительность
- Меньшая оптимизация
- Зависимость от интерпретатора
- Ошибки в процессе выполнения

ВИДЫ ЯП

- Гибридные ЯП



Примеры: Java, C#, Python

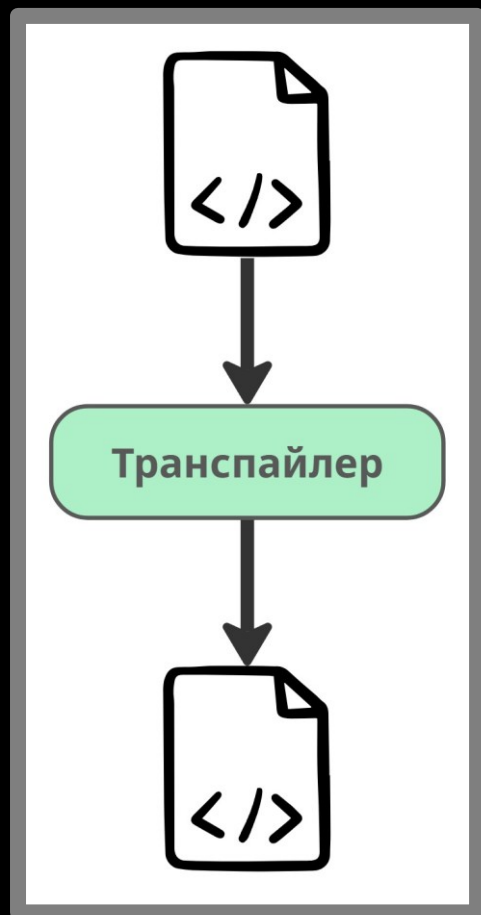
Плюсы:

- Баланс производительности и кроссплатформенности
- Проверка выполнения на уровне интерпретатора
- Динамическая оптимизация

Минусы:

- Дополнительный слой абстракции
- Зависимость от интерпретатора

- Транспайлер



Примеры: TypeScript -> JavaScript
CoffeeScript -> JavaScript

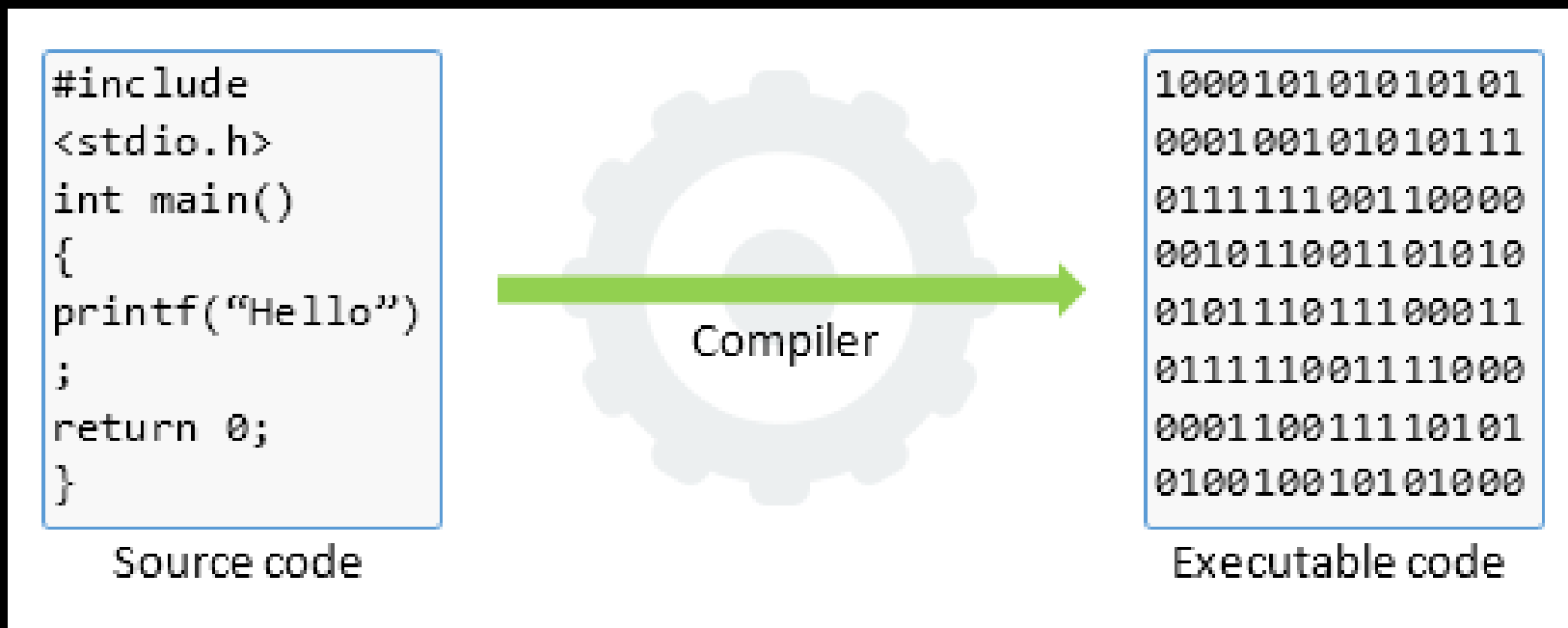
Плюсы:

- Совместимость
- Удобство разработки
- Быстрое внедрение стандартов

Минусы:

- Дополнительный этап сборки
- Зависимость от дополнительных инструментов

КОМПИЛЯТОР



Компилятор – программа, которая преобразует исходный код на языке программирования высокого уровня в машинный код

КОМПИЛЯТОР

Компилятор это ПО, на каком языке он написан?

- Обычно используется: asm, C, C++
- Может быть создан на том-же - самохостируемый компилятор

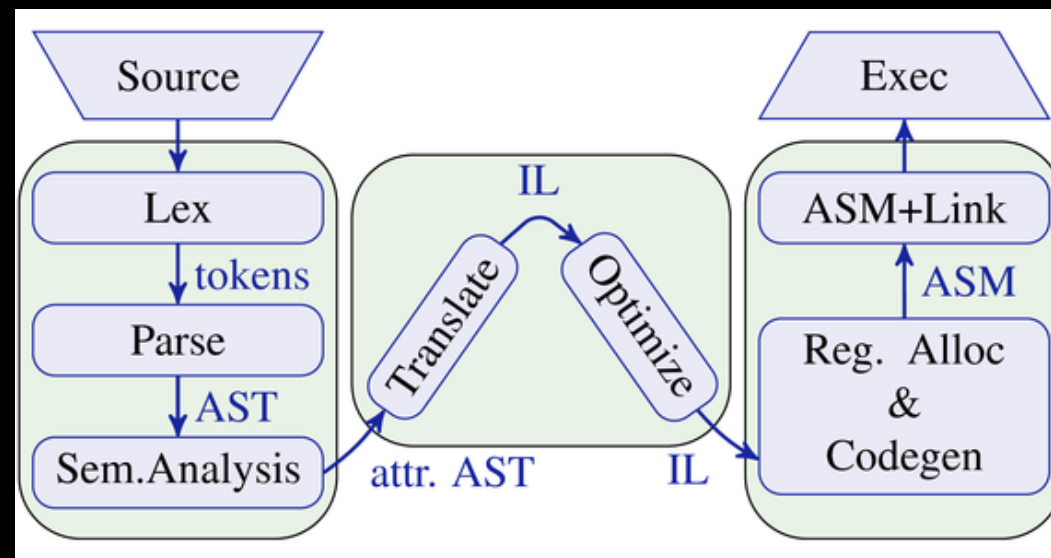
Примеры самохостируемых компиляторов:

- Для C: asm -> C
- Для Go: C -> Go

КОМПИЛЯТОР

Основные фазы компиляции программы:

1. Лексический анализ – разбиение на токены (лексемы)
2. Синтаксический анализ – дерево структуры программы
3. Семантический анализ – проверяет корректность между данными
4. Генератор промежуточного кода
5. Оптимизатор кода
6. Генератор целевого кода
7. Линковка библиотек



КОМПИЛЯТОР

1. Лексический анализ:

```
int main() {  
    int x = 5;  
    return x;  
}
```

=> разбивается на токены: `int`, `main`, `(`, `)`, `{`, `int`, `x`, `=`, `5`, `;`, `return`, `x`, `;`, `}`.

Этот процесс позволяет выявить синтаксические ошибки и подготовить код для дальнейшего анализа.

1. Пример лексического анализа:

```
const float factor = 42.f;
```

```
int calc(float x) {  
    return factor * x;  
}
```



```
(KW 'const'), (TYPE 'float'), (ID 'factor'),  
(EQ '='), (NUM '42.f'), (SEMI ';'), (TYPE 'int'),  
(ID 'calc'), (L_PAREN '('), (TYPE 'float'), (ID 'x')  
(R_PAREN ')'), (L_BRACE '{'), (KW 'return'),  
(ID 'factor'), (STAR '*'), (ID 'x'), (SEMI ';'),  
(R_BRACE '}'), (EOF '')
```

КОМПИЛЯТОР

2. Синтаксический анализ:

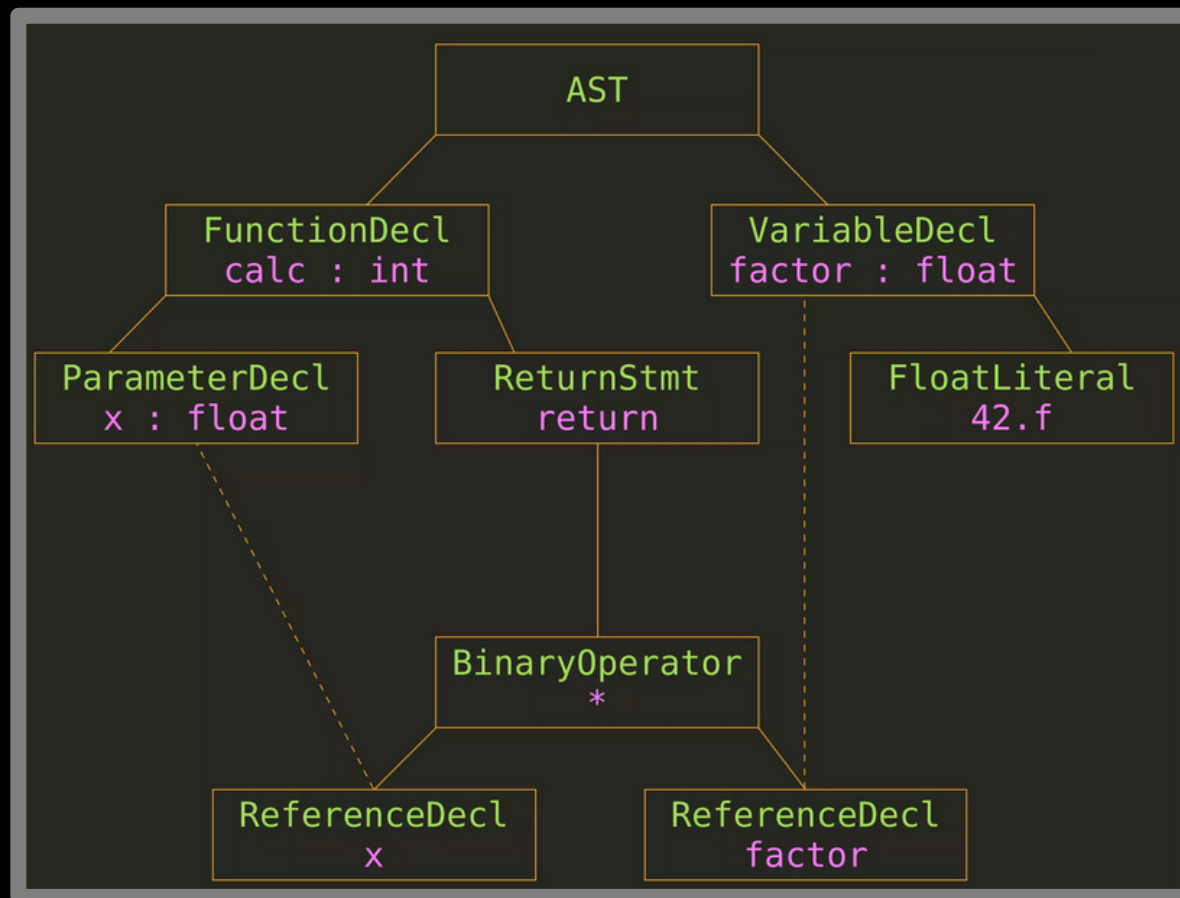
```
int main() {  
    int x = 5;  
    return x;  
}
```

=>

```
Program  
├─ FunctionDefinition (main)  
│   ├─ TypeSpecifier (int)  
│   ├─ Identifier (main)  
│   ├─ ParameterList (empty)  
│   └─ Block  
│       ├─ VariableDeclaration  
│       │   ├─ TypeSpecifier (int)  
│       │   ├─ Identifier (x)  
│       │   └─ Initializer (5)  
│       └─ ReturnStatement  
│           └─ Identifier (x)
```

Оценка соответствия
грамматики языка.

2. Пример синтаксического анализа:



3. Семантический анализ:

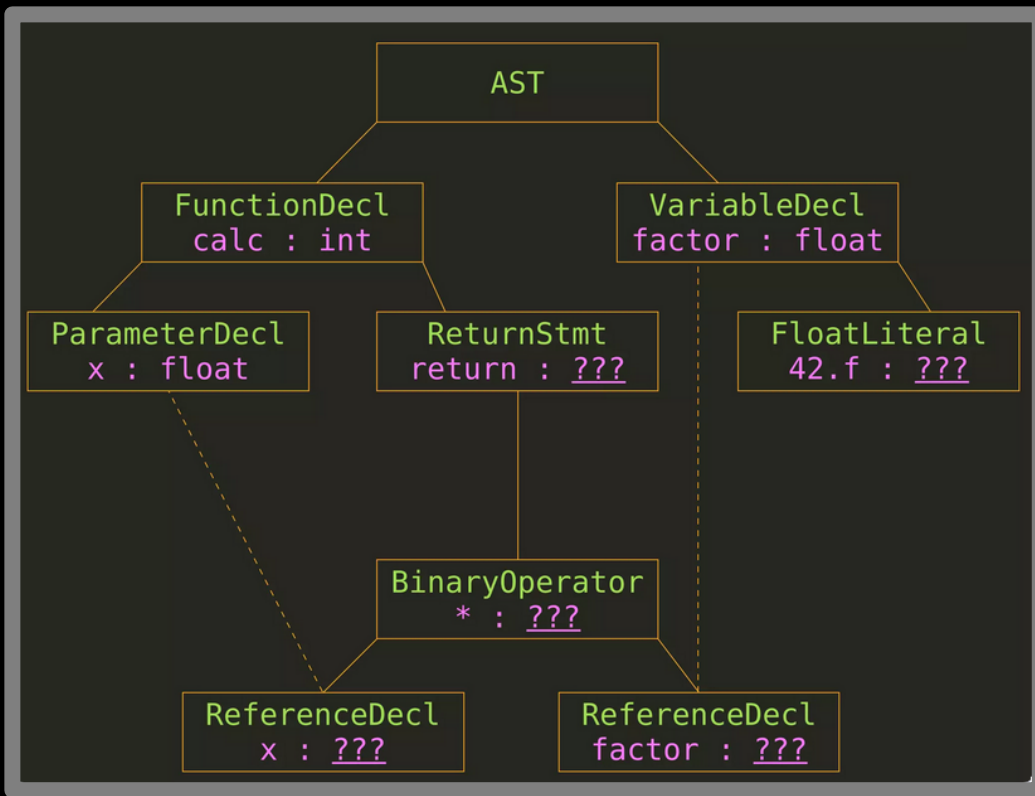
Проверка логической корректности кода.

Выявление:

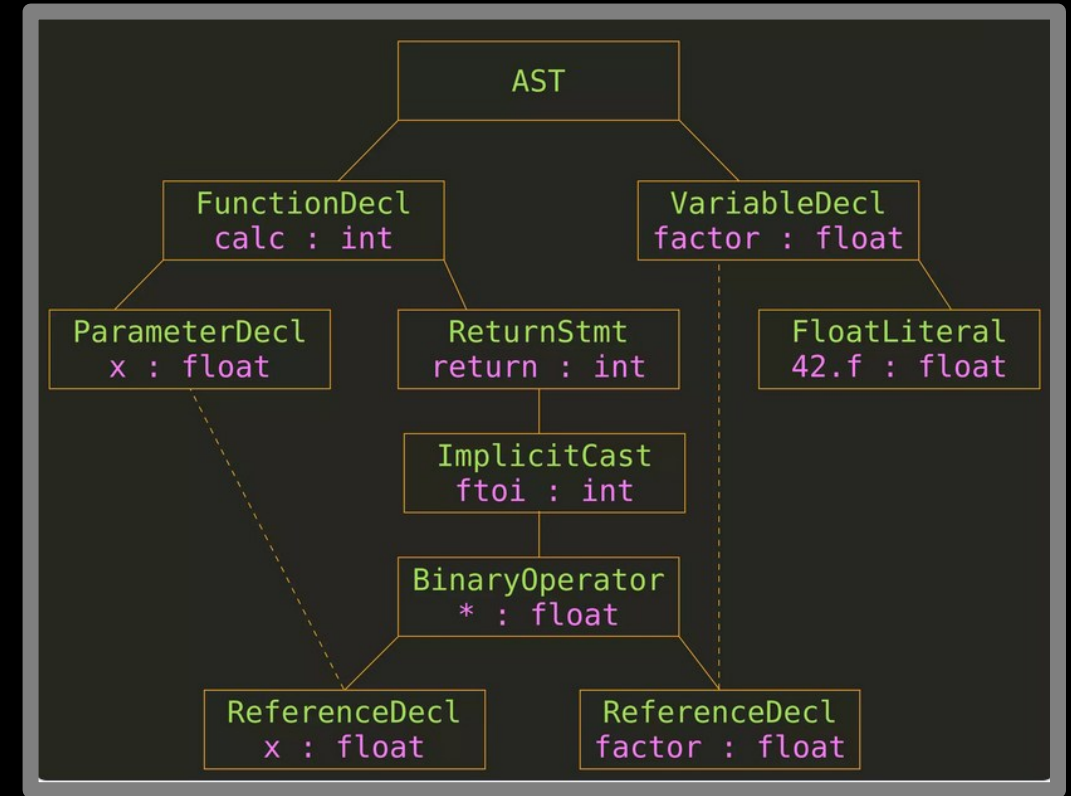
- Использование неопределенной переменной
- Несоответствие типов в операции
- Ошибки в разрешении имен
- Ошибки в использовании переменных и функций
- Ошибки в области видимости

КОМПИЛЯТОР

3. Примеры семантического анализа:



=>



4. Генератор промежуточного кода:

Преобразование кода в универсальное представление (например, трехадресный код, LLVM IR).

Назначение:

- Унификация
- Упрощение дальнейшего анализа
- Кроссплатформенность
- Разделение этапов компиляции

4. Пример генерации промежуточного кода:

```
@factor = constant float 42.0

define calc(float %x) {
entry:
    movf %x, %r1
    movf @factor, %r2
    %r3 = fmul %r1, %r2
    movf %r3, %r0
    ret
}
```

5. Оптимизатор кода:

Повышение производительности и уменьшение размера кода

Методы:

- Удаление мертвого кода
- Инлайнинг функций – замена вызова функций ее телом
- Раскрутка циклов – замена циклов с фиксированным числом итераций в последовательность команд
- Предсказание ветвлений – оптимизация условных переходов
- И пр.

5. Оптимизатор кода:

```
@factor = constant float 42.0
```

```
define calc(float %x) {  
entry:  
    movf %x, %r1  
    movf @factor, %r2  
    %r3 = fmul %r1, %r2  
    movf %r3, %r0  
    ret  
}
```

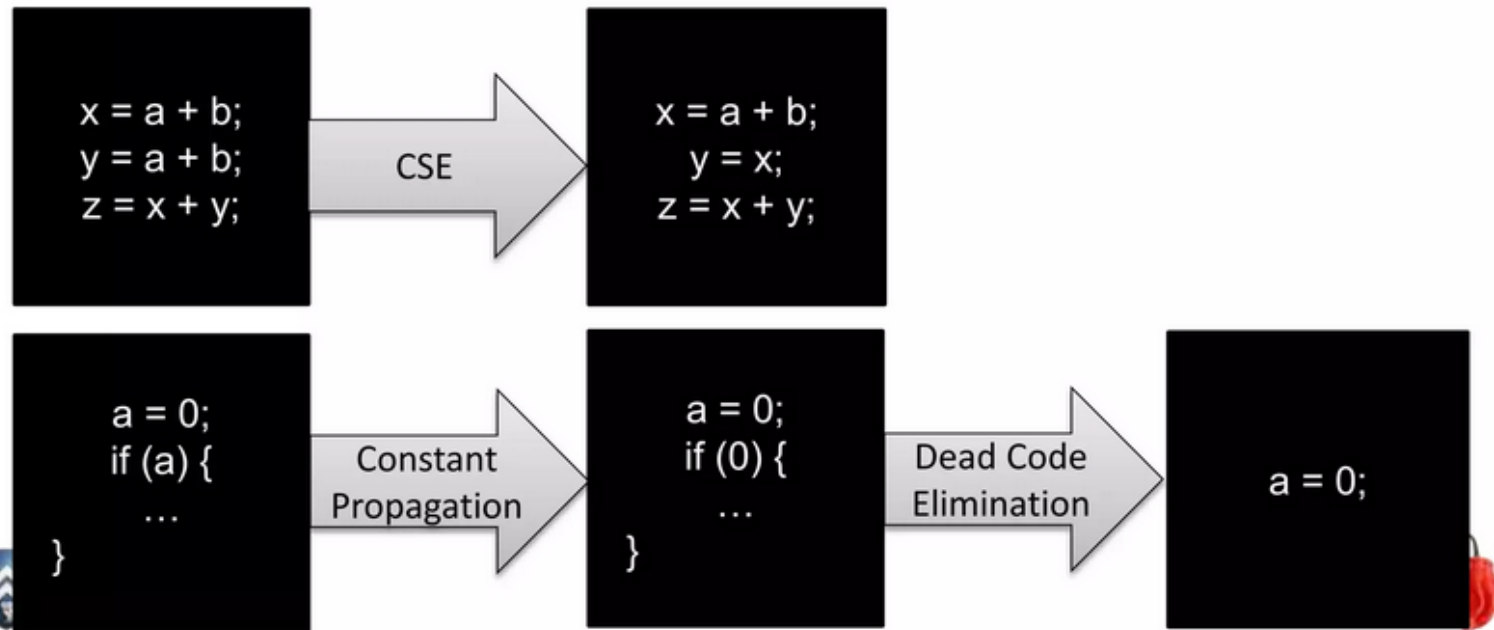
=>

```
@factor = constant float 42.0
```

```
define calc(float %x) {  
entry:  
    %r0 = fmul @factor, %x  
    ret  
}
```

5. Пример оптимизации кода:

Examples of optimizations: Scalar optimizations



5. Пример оптимизации кода:

Examples of optimizations: loop permutation (interchange)

Original

```
for (j = 0; j < N; j++) {  
  for (i = 0; i < M; i++) {  
    b[i][j] = a[i][j];  
  }  
}
```

Stride access
(slower on CPUs)

Interchanged

```
for (i = 0; i < M; i++) {  
  for (j = 0; j < N; j++) {  
    b[i][j] = a[i][j];  
  }  
}
```

Offset access
(faster on CPUs)



5. Пример оптимизации кода:

Examples of optimizations: loop fusion/distribution

Fused

```
for (i = 0; i < N; i++) {  
    a[i] = b[i] + c[i];  
    d[i] = a[i] + e[i];  
}
```

Better temporal locality
on CPUs

Distributed

```
for (i = 0; i < N; i++) {  
    a[i] = b[i] + c[i];  
}  
for (i = 0; i < N; i++) {  
    d[i] = a[i] + e[i];  
}
```

Good for Vectorization
on CPUs



Depending on the loop size "N"

13



6. Генератор целевого кода:

Промежуточный код преобразуется в машинные инструкции (для определенной ОС и архитектуры).

```
t1 = 5  
t2 = 3  
t3 = t1 + t2  
x = t3
```

=>

```
b8 04 00 00 00  
bb 01 00 00 00  
b9 00 00 00 00
```

КОМПИЛЯТОР

6. Пример генерации целевого кода:

```
@factor = constant float 42.0
```

```
define calc(float %x) {  
  entry:  
    %r0 = fmul @factor, %x  
    ret  
}
```

=>

```
_calc:  
  push {r7, lr}  
  mov r7, sp  
  mov r1, #36175872  
  orr r1, r1, #1073741824  
  bl ___mulsf3  
  bl ___fixsfsi  
  pop {r7, lr}  
  mov pc, lr  
  
  .section __TEXT,__const  
  .globl _factor @ @factor  
  .align 2  
_factor:  
  .long 1109917696 @ float 42
```

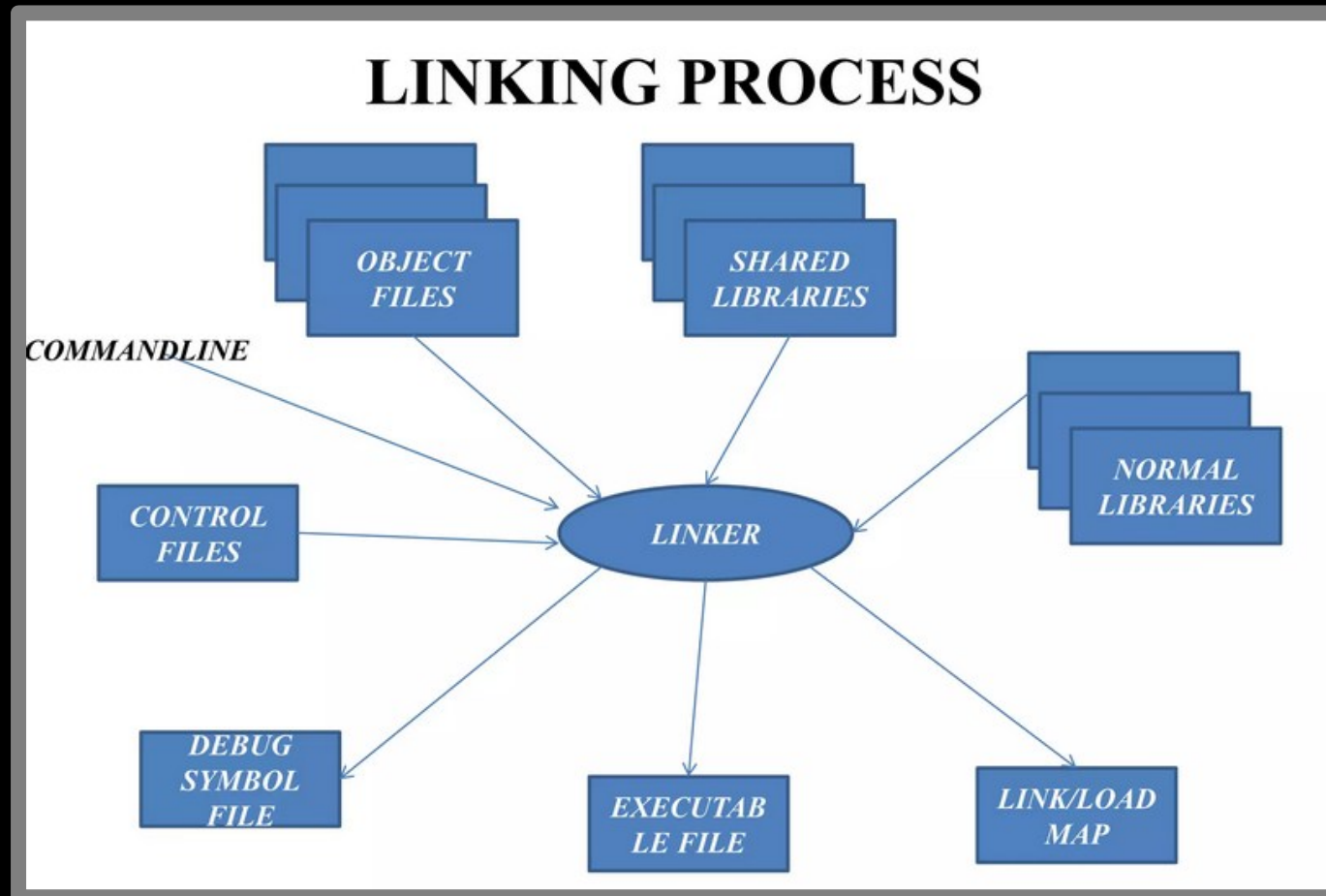

КОМПИЛЯТОР

7. Линковка библиотек:

Компоновщик объединяет несколько объектных файлов в исполняемый файл, разрешая внешние ссылки между ними.

Линковка может быть статической (все зависимости включаются в исполняемый файл) или динамической (использование библиотек во время выполнения).

7. Линковка библиотек:



7. Пример линковки библиотек:

```
extern int calc(float);

int main() {
    printf("%d\n", calc(2.f));
    return 0;
}
```

```
> clang -c main.c -o main.o
```

=>

```
> nm main.o
```

```
                                U _calc
00000000000000000000 T _main
                                U _printf
```

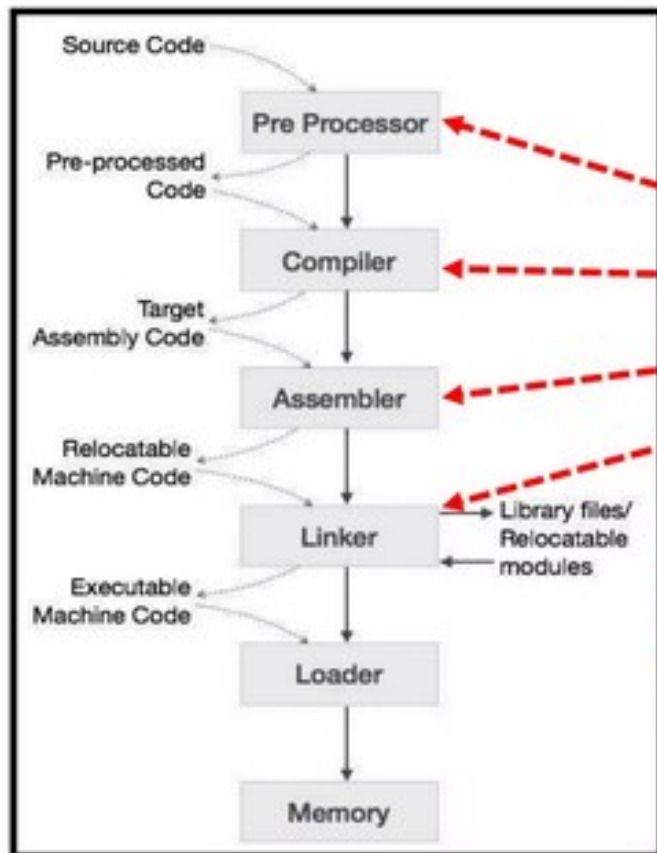
```
> ld -lc calc.o main.o -o main
```

```
> nm main
```

```
00000000000000001f30 T _calc
00000000000000001fc8 S _factor
00000000000000001f60 T _main
                                U _printf
```


КОМПИЛЯТОР

Пример поэтапной компиляции:



C:\Windows\System32\cmd.exe

C:\Users\AHmed.ELGhafar\Desktop\Lec_3>cpp application.c > application.i

C:\Users\AHmed.ELGhafar\Desktop\Lec_3>gcc -S application.i

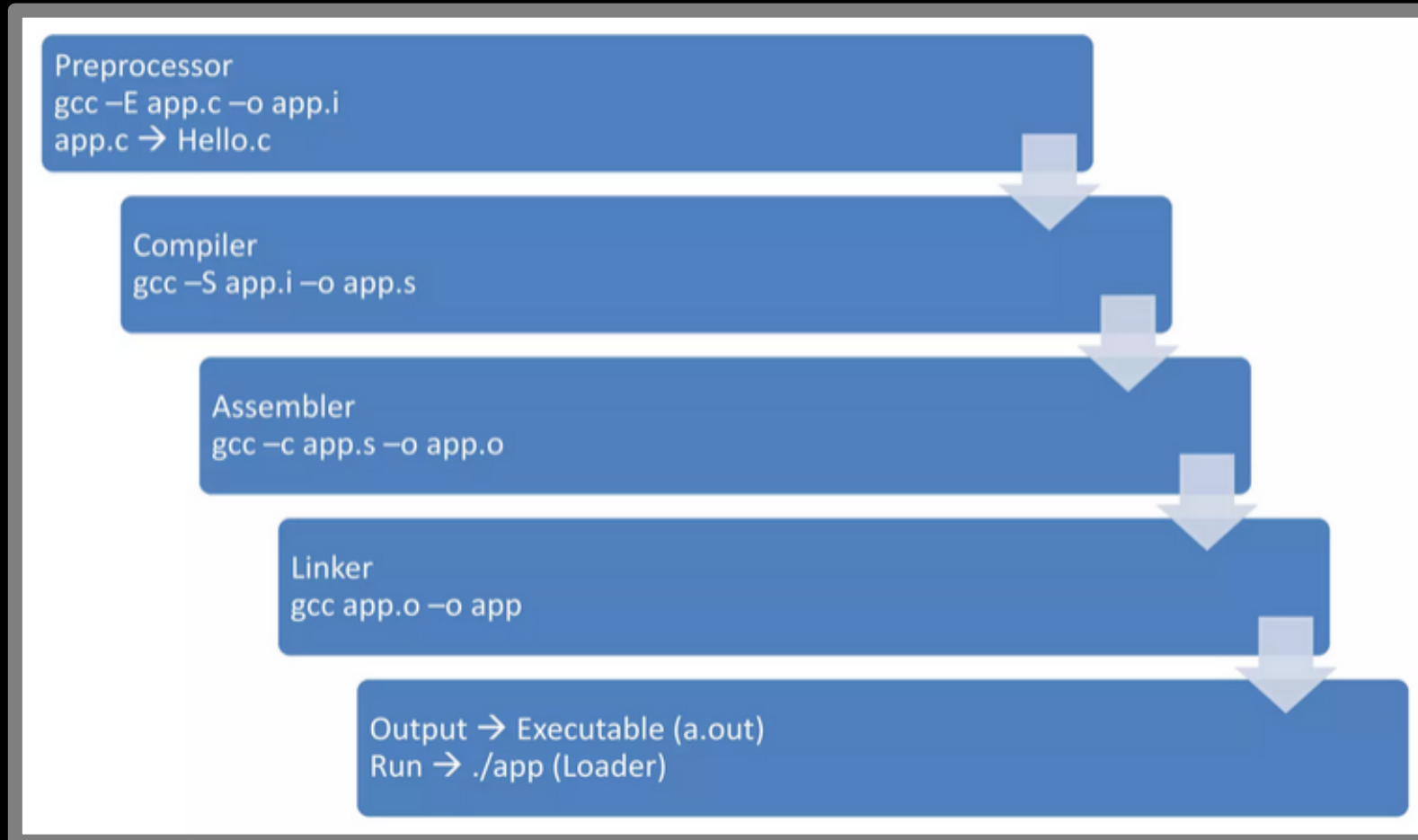
C:\Users\AHmed.ELGhafar\Desktop\Lec_3>as -o application.o application.s

C:\Users\AHmed.ELGhafar\Desktop\Lec_3>ld -o application.exe application.o -L.
application.o:application.c:(.text+0x9): undefined reference to `__main'
application.o:application.c:(.text+0x27): undefined reference to `printf'
application.o:application.c:(.text+0x50): undefined reference to `puts'
application.o:application.c:(.text+0x6b): undefined reference to `puts'

C:\Users\AHmed.ELGhafar\Desktop\Lec_3>

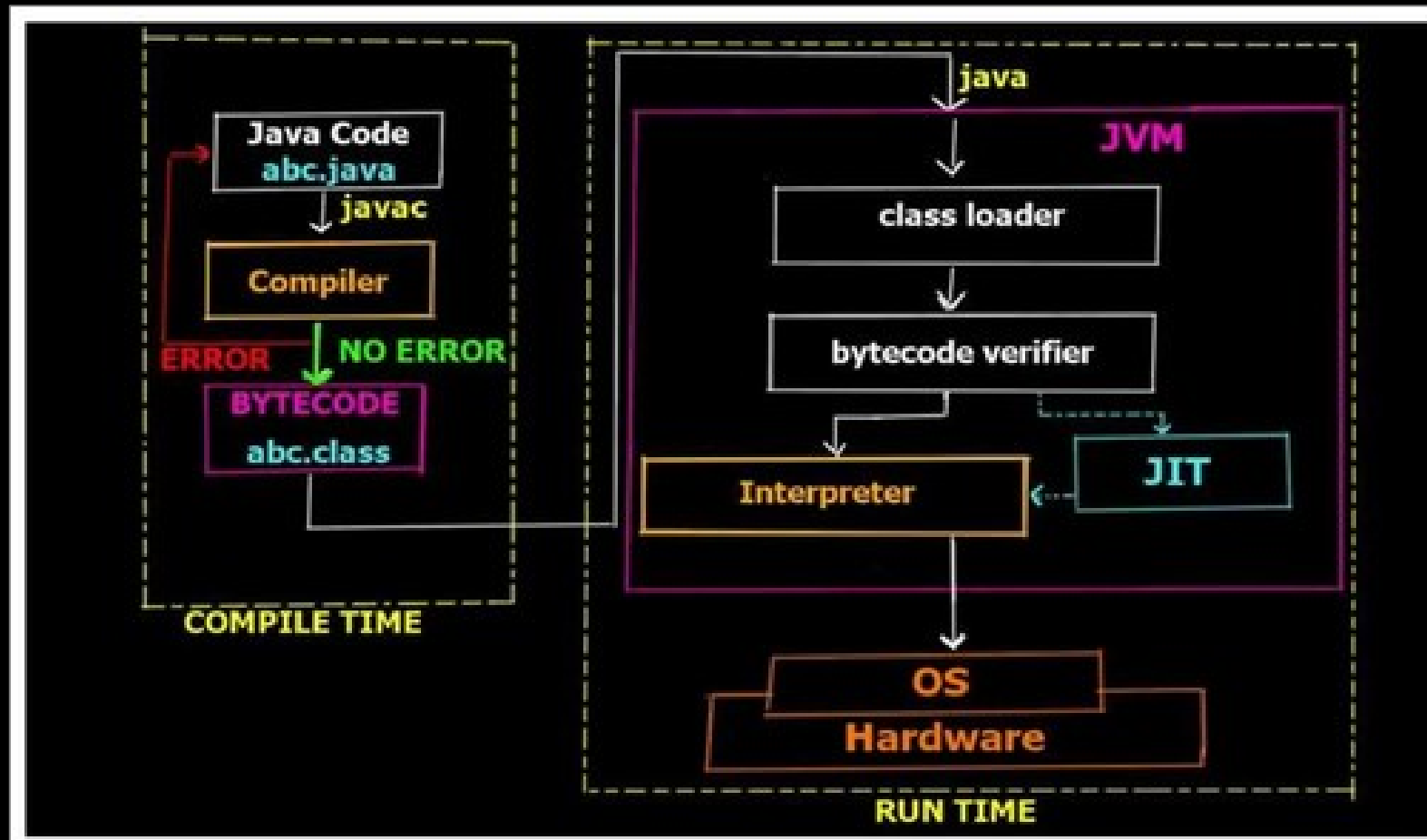
КОМПИЛЯТОР

Пример поэтапной компиляции:



КОМПИЛЯТОР

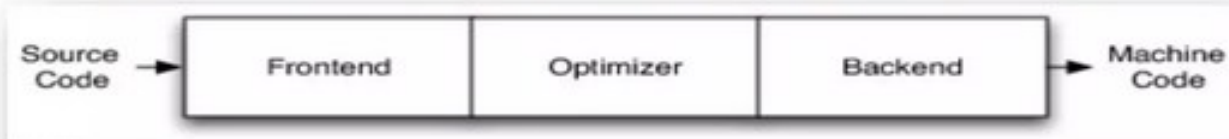
JAVA:



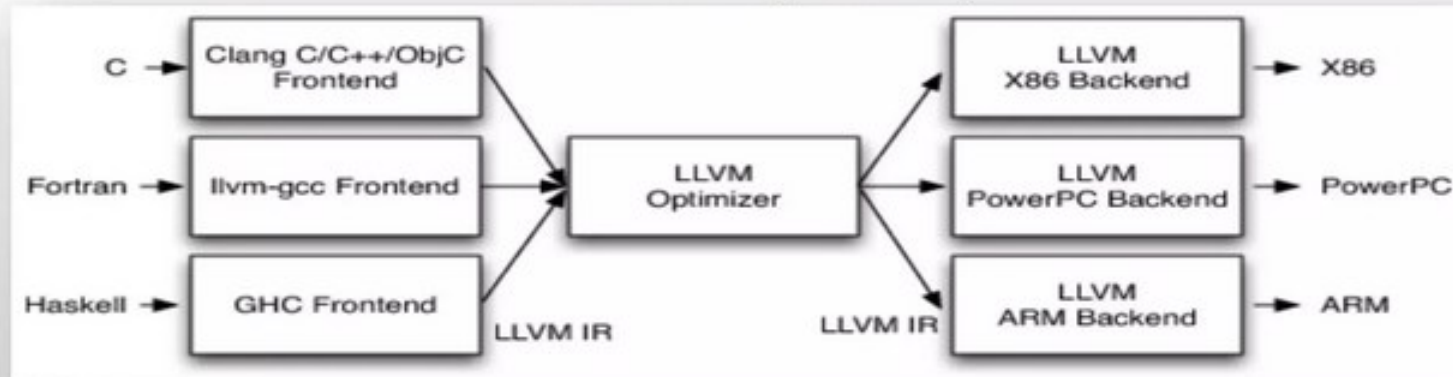
КОМПИЛЯТОР

LLVM – платформа для создания компиляторов:

Traditional Three-Phase Compiler design



LLVM retargetability



КОМПИЛЯТОР

Потенциальное влияние компилятора на безопасность.

Компиляторы могут вносить уязвимости на этапах: лексического, синтаксического, семантического анализа и оптимизации кода.

Примеры:

- Инъекция кода
- Удаление части функций (например, проверки данных)
- Неопределенное поведение
- Небезопасное использование инструкций процессора
- Ошибки в байт-коде могут привести к некорректному выполнению кода или утечки памяти

КОМПИЛЯТОР

Встроенные средства защит в компилятор:

- ASan (AddressSanitizer) – обращение к невыделенной памяти
- UBSan (UndefinedBehaviorSanitizer) – неопределенное поведение
- CFI (Control-Flow Integrity) – целостность потока управления
- DEP (Data Execution Prevention) – предотвращение выполнение данных
- ASLR (Address Space Layout Randomization) – усложнение предсказуемость адресов
- Borrow-checker – безопасность памяти
- И пр.



ГОСТ БРПО

П.5.12 Использование безопасной системы сборки ПО

Цель:

- Безопасная сборка ПО
- Недопущение внесения в код ошибок

Требования к реализации:

- Регламент сборки ПО
- Информация о системе сборки ПО и среды
- Обеспечить выполнение рекомендаций производителя системы сборки ПО

Регламент сборки ПО:

- Обязанности сотрудников и их роли
- Критерии выбора инструментов
- Критерии приемки результатов
- Регистрация событий сборки

Информация о сборочной среде:

- Описание функционирования сборочной среды
- Перечень программных инструментов (версия, конфигурация)



РЕКОМЕНДАЦИИ

РЕКОМЕНДАЦИИ

Использовать доверенный компилятор.

- Использовать доверенный ресурс (официальный), проверить целостность или цифровую подпись
- Собрать из исходников (проблема курицы и яйца)
- Отслеживать версию компиляторов и его окружение
- Компилируйте код в изолированной среде

РЕКОМЕНДАЦИИ

Обращать внимание на критические опции и опции оптимизации:

Warning Options

```
-fsyntax-only -fmax-errors=n -Wpedantic -pedantic-errors -w
-Wextra -Wall -Waddress -Waddress-of-packed-member
-Waggregate-return -Waligned-new -Walloc-zero
-Walloc-size-larger-than=byte-size -Walloca
-Walloca-larger-than=byte-size
-Wno-aggressive-loop-optimizations -Warray-bounds
-Warray-bounds=n -Wno-attributes -Wattribute-alias=n
-Wbool-compare -Wbool-operation
-Wno-builtin-declaration-mismatch -Wno-builtin-macro-redefined
-Wc90-c99-compat -Wc99-c11-compat -Wc11-c2x-compat
-Wc++-compat -Wc++11-compat -Wc++14-compat -Wc++17-compat
-Wcast-align -Wcast-align=strict -Wcast-function-type
-Wcast-qual -Wchar-subscripts -Wcatch-value -Wcatch-value=n
-Wclobbered -Wcomment -Wconditionally-supported -Wconversion
-Wcoverage-mismatch -Wno-cpp -Wdangling-else -Wdate-time
-Wdelete-incomplete -Wno-attribute-warning -Wno-deprecated
-Wno-deprecated-declarations -Wno-designated-init
-Wdisabled-optimization -Wno-discarded-qualifiers
-Wno-discarded-array-qualifiers -Wno-div-by-zero
-Wdouble-promotion -Wduplicated-branches -Wduplicated-cond
-Wempty-body -Wenum-compare -Wno-endif-labels
-Wexpansion-to-defined -Werror -Werror=* -Wextra-semi
-Wfatal-errors -Wfloat-equal -Wformat -Wformat=2
-Wno-format-contains-nul -Wno-format-extra-args
-Wformat-nonliteral -Wformat-overflow=n -Wformat-security
-Wformat-signedness -Wformat-truncation=n -Wformat-y2k
-Wframe-address -Wframe-larger-than=byte-size
-Wno-free-nonheap-object -Wjump-misses-init -Whsa
-Wif-not-aligned -Wignored-qualifiers -Wignored-attributes
-Wincompatible-pointer-types -Wimplicit
```

Optimization Options

```
-faggressive-loop-optimizations
-falign-functions[=n[:m[:n2[:m2]]]]
-falign-jumps[=n[:m[:n2[:m2]]]]
-falign-labels[=n[:m[:n2[:m2]]]]
-falign-loops[=n[:m[:n2[:m2]]]] -fassociative-math
-fauto-profile -fauto-profile[=path] -fauto-inc-dec
-fbranch-probabilities -fbranch-target-load-optimize
-fbranch-target-load-optimize2 -fbtr-bb-exclusive
-fcaller-saves -fcombine-stack-adjustments -fconserve-stack
-fcompare-elim -fcpop-registers -fcrossjumping
-fcse-follow-jumps -fcse-skip-blocks -fcx-fortran-rules
-fcx-limited-range -fdata-sections -fdce -fdelayed-branch
-fdelete-null-pointer-checks -fdevirtualize
-fdevirtualize-speculatively -fdevirtualize-at-ltrans -fdse
-fearly-inlining -fipa-sra -fexpensive-optimizations
-ffat-lto-objects -ffast-math -ffinite-math-only
-ffloat-store -fexcess-precision=style -fforward-propagate
-ffp-contract=style -ffunction-sections -fgcse
-fgcse-after-reload -fgcse-las -fgcse-lm
-fgraphite-identity -fgcse-sm -fhoist-adjacent-loads
-fif-conversion -fif-conversion2 -findirect-inlining
-finline-functions -finline-functions-called-once
-finline-limit=n -finline-small-functions -fipa-cp
-fipa-cp-clone -fipa-bit-cp -fipa-vrp -fipa-pta
-fipa-profile -fipa-pure-const -fipa-reference
-fipa-reference-addressable -fipa-stack-alignment -fipa-icf
-fira-algorithm=algorithm -flive-patching=level
-fira-region=region -fira-hoist-pressure -fira-loop-pressure
-fno-ira-share-save-slots -fno-ira-share-spill-slots
-fisolate-erroneous-paths-dereference
```

РЕКОМЕНДАЦИИ

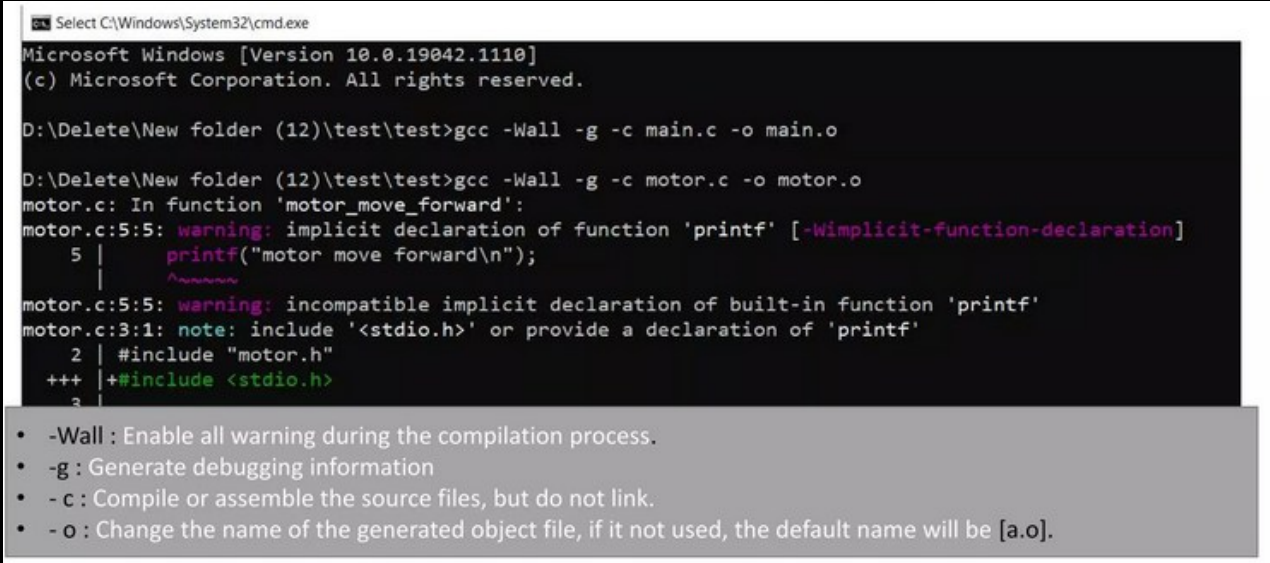
Примеры для gcc:

- `-fno-stack-protector` – отключает защиту стека от переполнения
- `-fno-pie`, `-fno-pic` – отключает механизм PIE, PIC
- `-W1`, `--disable-new-dtags`, `--as-needed` – отключить механизм защиты динамического линковщика
- `-z execstack` – делает стек исполняемым
- `-O0` – отсутствие оптимизации, которая удаляет может удалять небезопасные конструкции
- `-fcommon` – разрешает множественное определение глобальных переменных
- `-fpermissive` – позволяет компилировать код с нарушением стандарта

РЕКОМЕНДАЦИИ

Используйте журналирование сборки:

- Собирайте журналы работы компилятора (включая события Warning)
- Отслеживайте системные вызовы компилятора (strace) на предмет обращения к сторонним объектам (файлам, открытие сокетов)



```
Select C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19042.1110]
(c) Microsoft Corporation. All rights reserved.

D:\Delete\New folder (12)\test\test>gcc -Wall -g -c main.c -o main.o

D:\Delete\New folder (12)\test\test>gcc -Wall -g -c motor.c -o motor.o
motor.c: In function 'motor_move_forward':
motor.c:5:5: warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
   5 |     printf("motor move forward\n");
     |     ~~~~~
motor.c:5:5: warning: incompatible implicit declaration of built-in function 'printf'
motor.c:3:1: note: include '<stdio.h>' or provide a declaration of 'printf'
   2 | #include "motor.h"
   +++ |+#include <stdio.h>
   3 |
```

- -Wall : Enable all warning during the compilation process.
- -g : Generate debugging information
- -c : Compile or assemble the source files, but do not link.
- -o : Change the name of the generated object file, if it not used, the default name will be [a.o].

РЕКОМЕНДАЦИИ

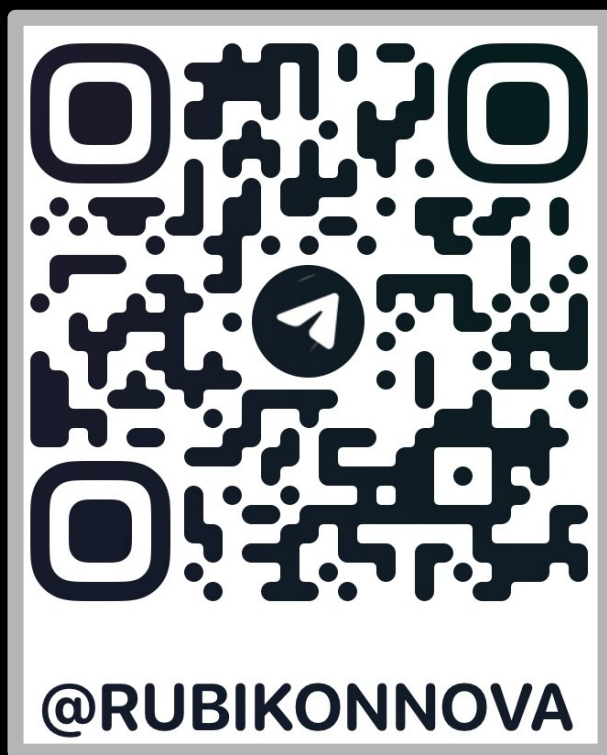
Проверяйте собранные файлы:

- Сборка исходных файлов должна быть воспроизводима
- Исследование используемых динамических библиотек (nm, ldd)
- Исследование содержимого исполняемого файла (objdump)
- Просмотр двоичного файла (xdd)

Реверс-инжиниринг (CTF-практика):

- IDA
- Radare2
- Ghidra

Спасибо за внимание



НПО Эшелон

<https://npo-echelon.ru>

<https://t.me/RubikonNova>