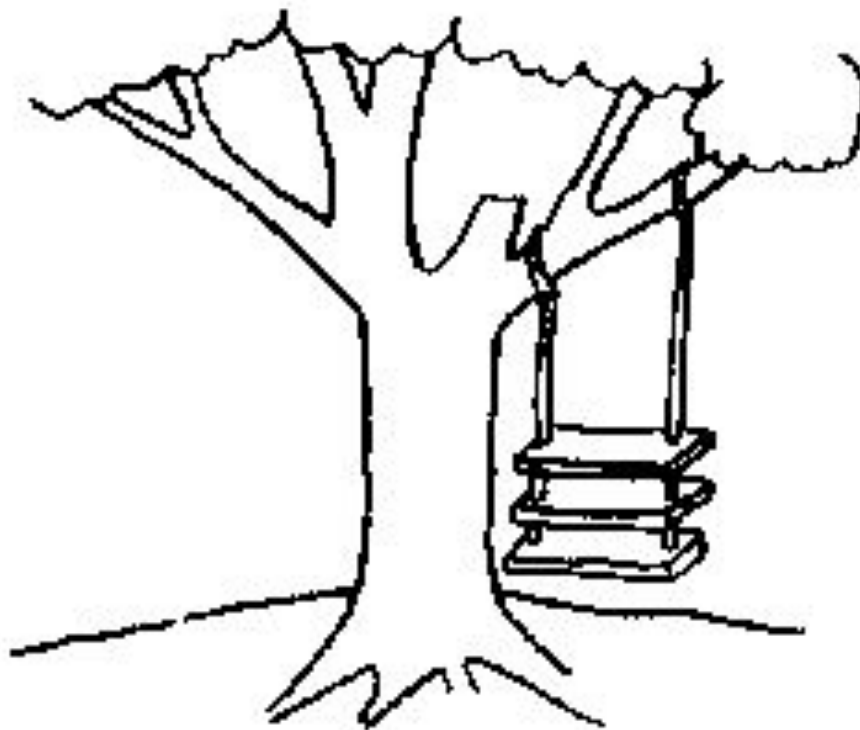
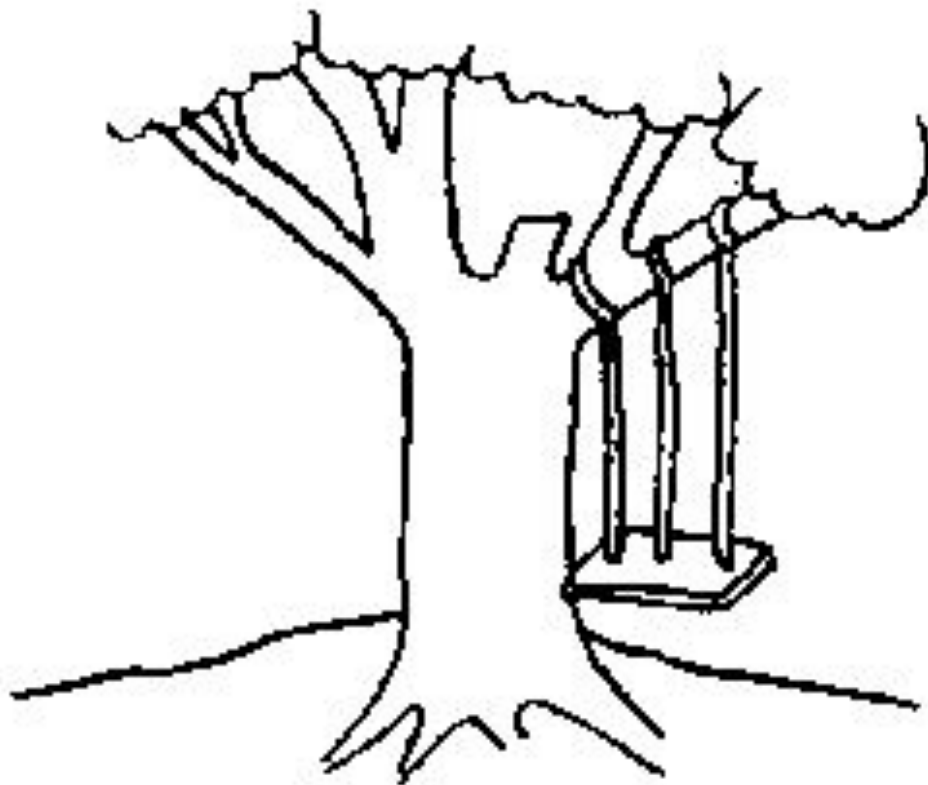

Вычислительная техника и компьютерное моделирование в физике

Зинчик Александр Адольфович
zinchik_alex@mail.ru

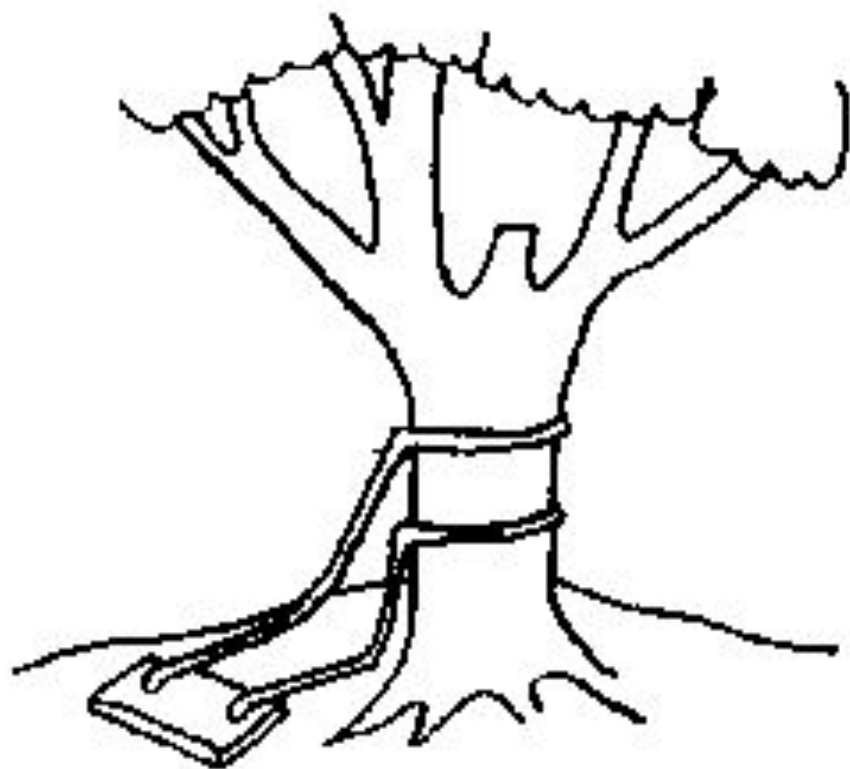
«Качели» - как проектируются программы :)



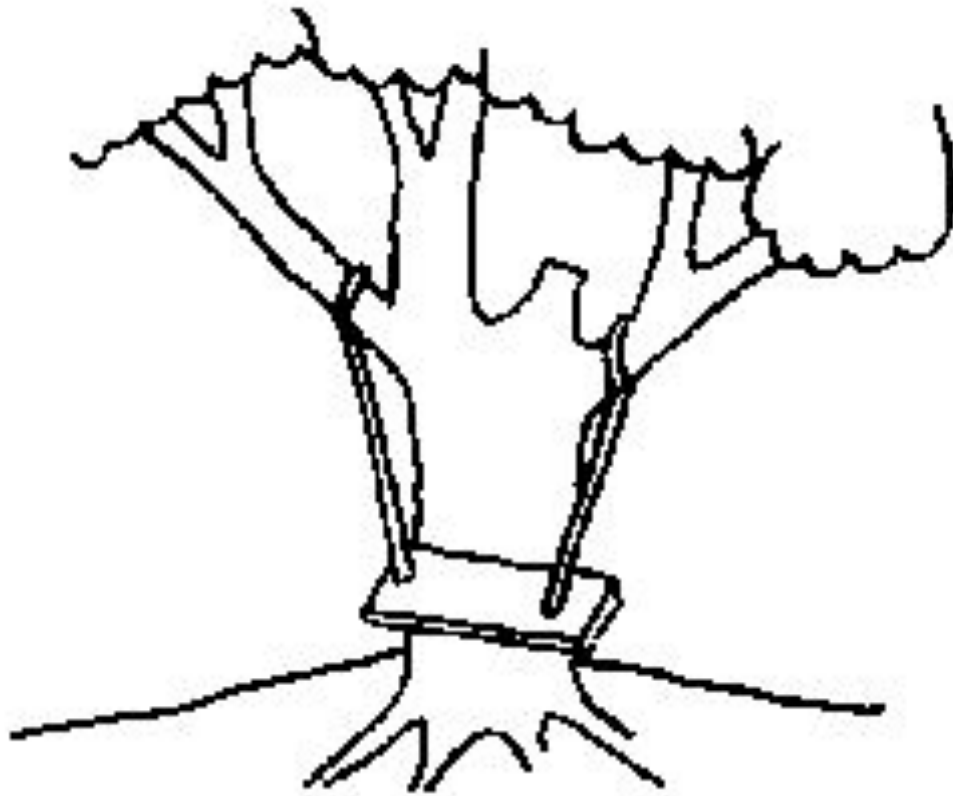
**Как было предложено
организатором разработки**



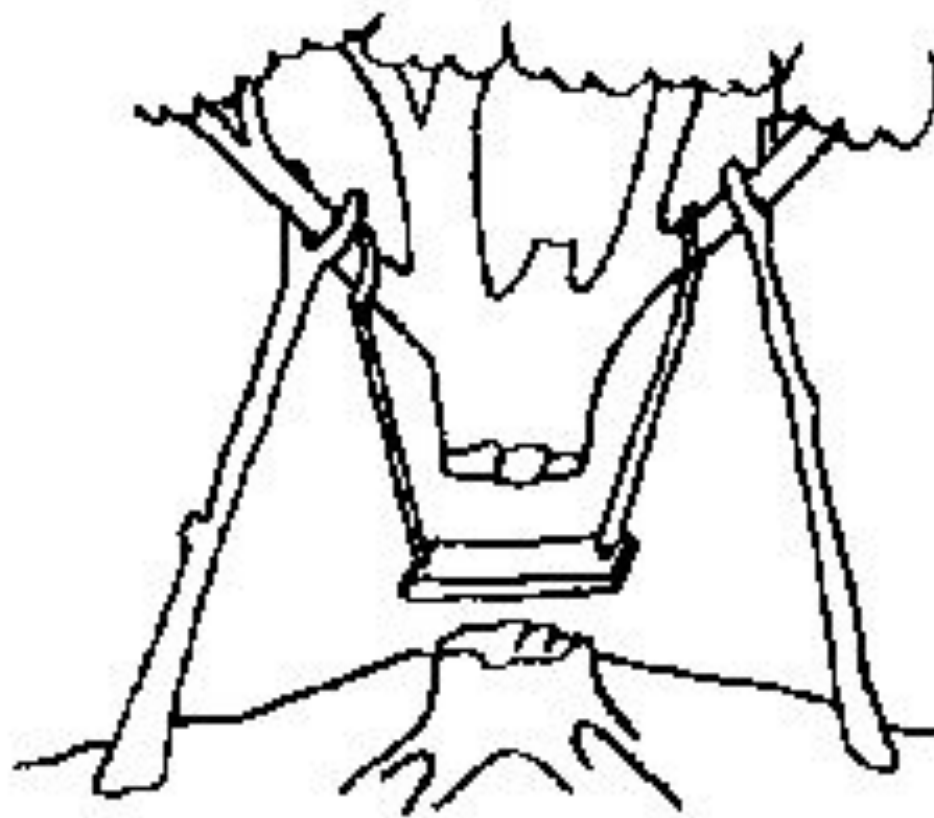
**Как было описано
в техническом задании**



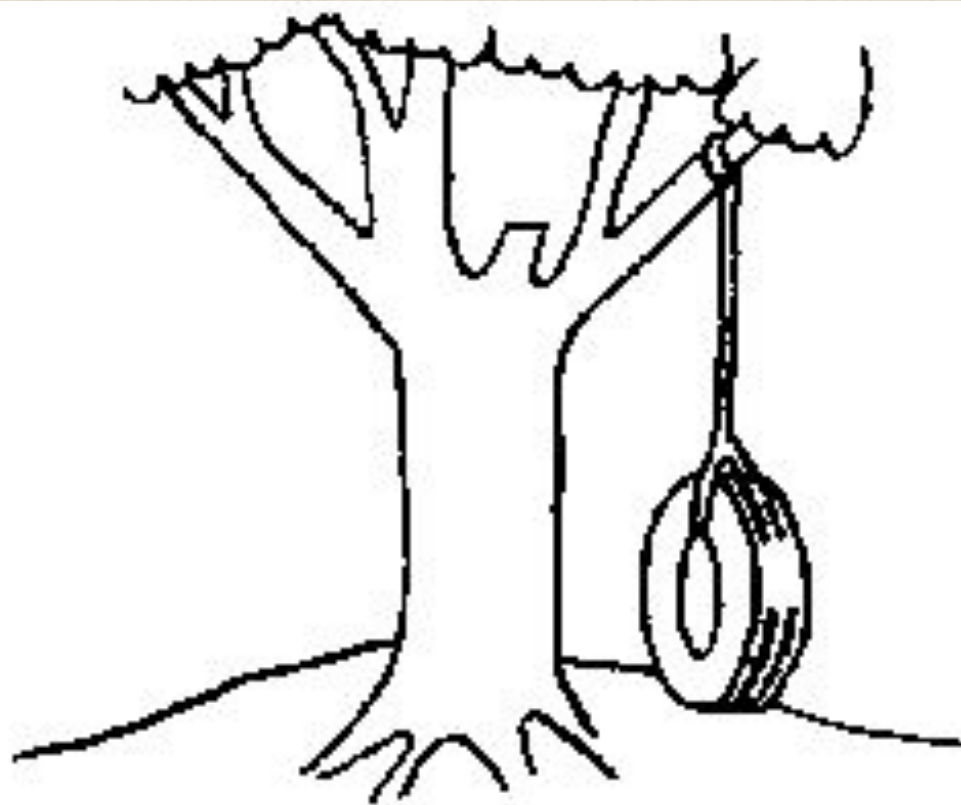
**Как было спроектировано
ведущим системным специалистом**



**Как было реализовано
программистами**

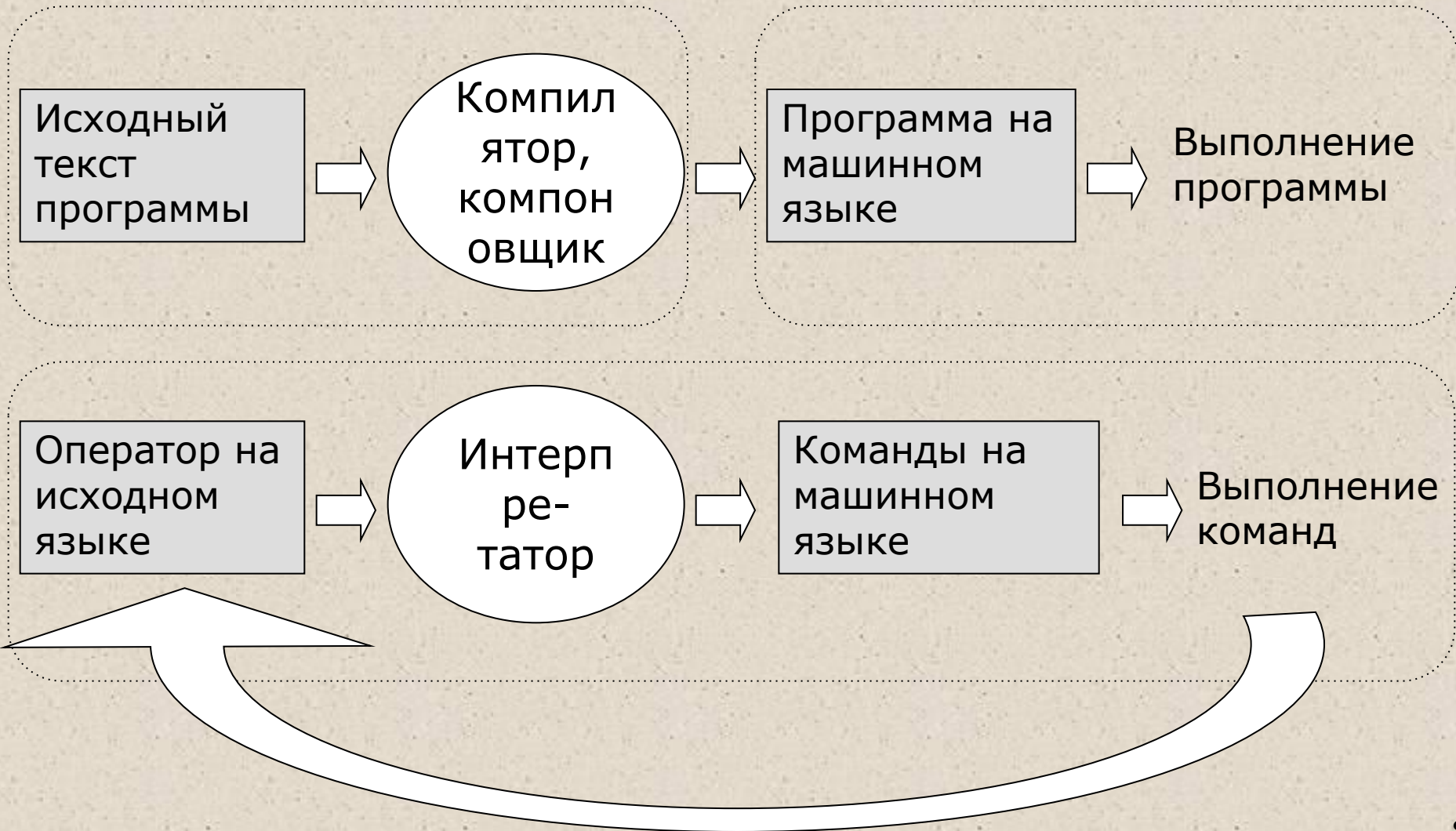


Как было внедрено



Чего хотел пользователь

Компилятор или интерпретатор



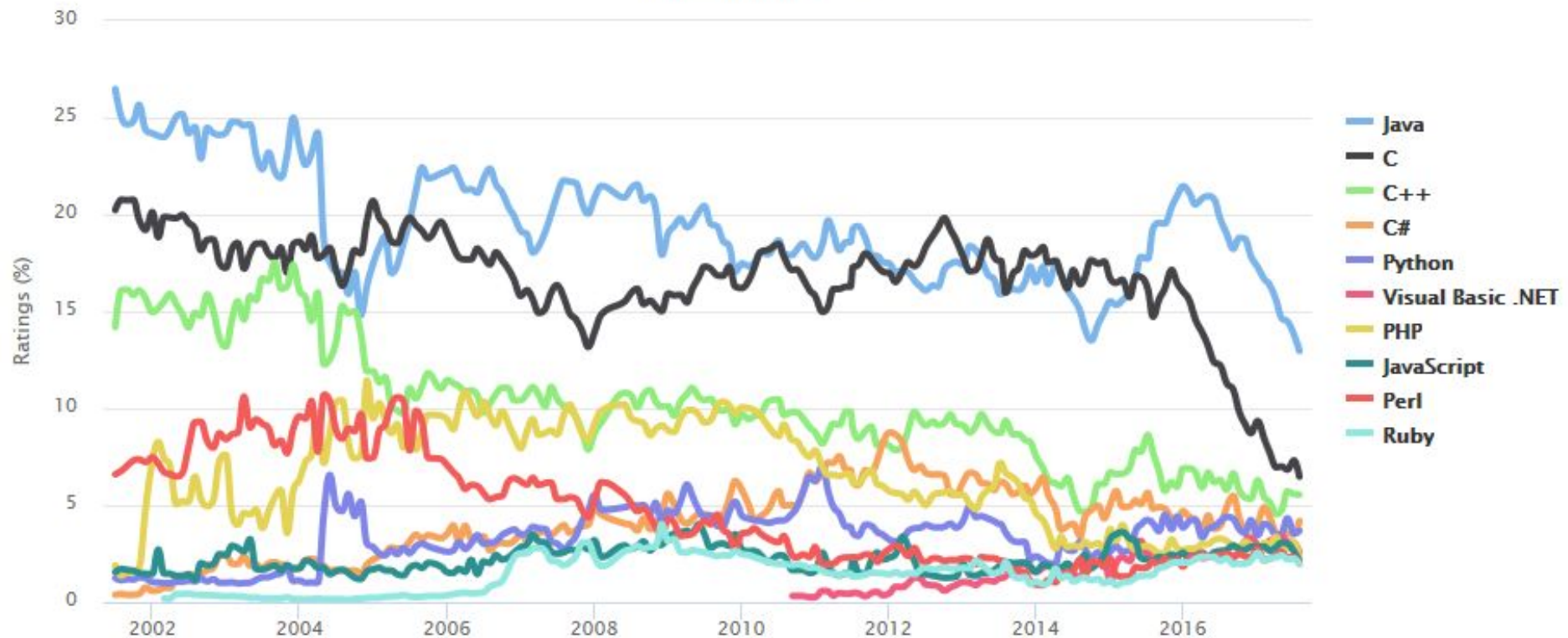
Достоинства и недостатки

КОМПИЛИРУЕМЫЙ		ИНТЕРПРЕТИРУЕМЫЙ	
ГОТОВ К ЗАПУСКУ	НЕ КРОСС-ПЛАТФОРМЕННЫЙ	КРОСС-ПЛАТФОРМЕННЫЙ	ПОСТОЯННО интерпретировать
РАБОТАЕТ БЫСТРЕЕ	НЕГИБКИЙ	ПРОЩЕ ТЕСТИРОВАТЬ	работает медленнее
СКРЫТ ИСХОДНЫЙ КОД	ДОПОЛНИТЕЛЬНЫЕ ШАГИ	проще отлаживать	доступен исходный код

Рейтинг популярности языков программирования

TIOBE Programming Community Index

Source: www.tiobe.com



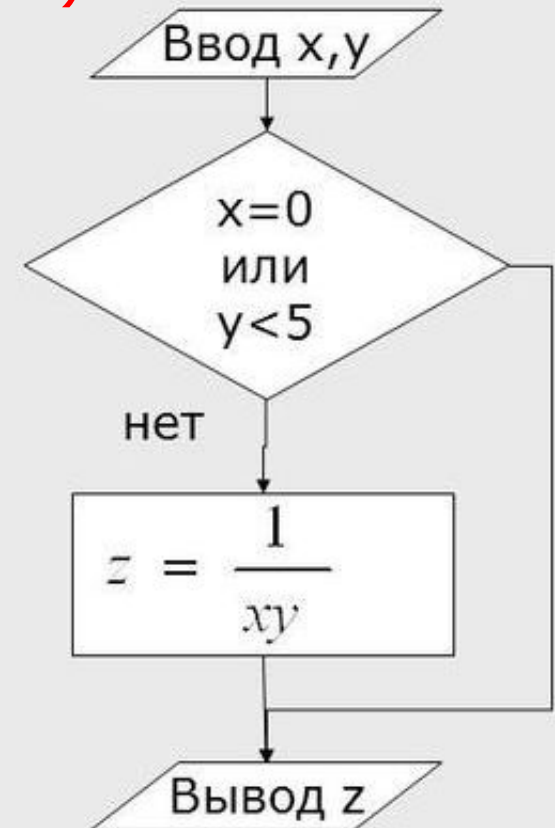
Небольшой тест

1) В двоичном коде число 15.625 запишется как...

2) Что выведет следующая программа

```
float a, b, c;  
a=1e20;  
b=1e10;  
c = a + b;  
if(a==c) cout<<"Equal"<<endl;  
else cout<<"Not equal"<<endl;
```

3)



3) Напишите программу, соответствующую приведенной схеме алгоритма, на любом известном вам языке программирования. Все переменные вещественные.

Критерии качества ПО

Внешние характеристики

- корректность
- практичность
- эффективность
- надежность
- целостность
- адаптируемость
- ...

Внутренние характеристики

- удобство
сопровождения
- тестируемость
- удобочитаемость
- гибкость
- портируемость
- ...

Основные критерии качества программы

- надежность
- возможность точно планировать производство и сопровождение
Для достижения этих целей программа должна:
- иметь простую структуру
- быть хорошо читаемой
- быть легко модифицируемой

Парадигмы программирования

- Парадигма — способ организации программы, то есть принцип ее построения. Наиболее распространенными являются процедурная и объектно-ориентированная парадигмы.

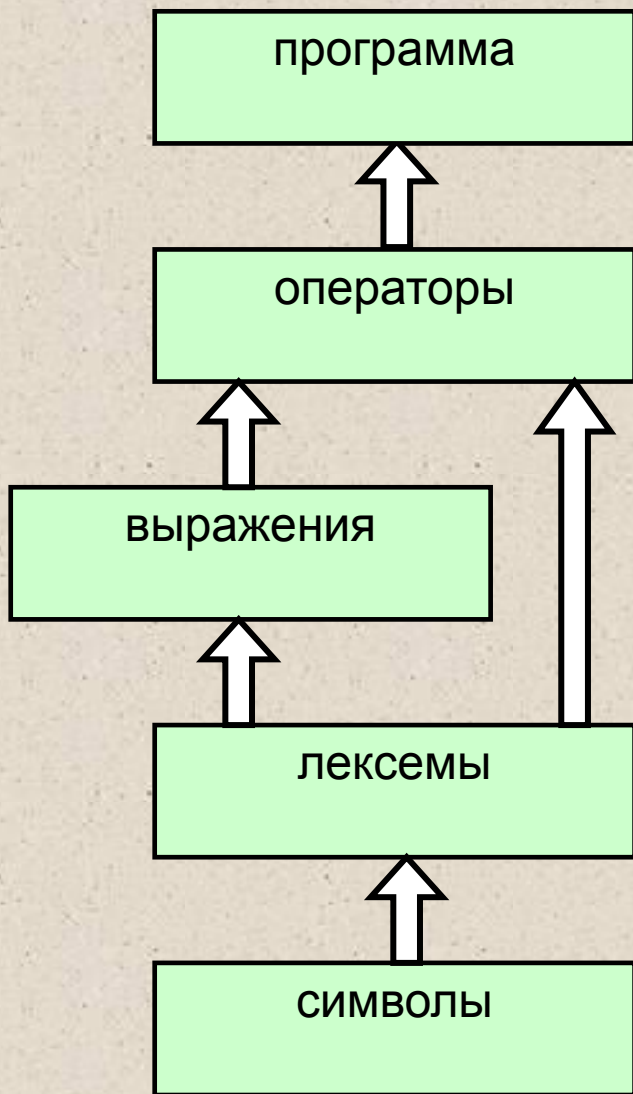
Они различаются способом декомпозиции, положенным в основу при создании программы.

- *Процедурная декомпозиция* состоит в том, что задача, реализуемая программой, делится на подзадачи, а они, в свою очередь — на более мелкие этапы, то есть выполняется пошаговая детализация алгоритма решения задачи.
- *Объектно-ориентированная декомпозиция* предполагает разбиение предметной области на объекты и реализацию этих объектов и их взаимосвязей в виде программы.

Базовые средства C++

- Типы данных C++
- Структура программы
- Переменные и выражения
- Базовые конструкции структурного программирования (операторы ветвления, цикла и т.д.)
- Указатели и ссылки
- Массивы, строки
- Типы данных, определяемые пользователем (enum, struct, union)

Состав языка



`a=b; for (int i=0;i<n;++i)`

`a++ - --b/c`

- идентификаторы
- ключевые слова
- константы
- знаки операций
- разделители

`a-z, A-Z, 0-9, " , {,},|,/,%,...`

примеры

Пример структуры программы

директивы препроцессора

описания

```
int main() {
```

операторы главной функции

```
}
```

```
int f1() {
```

операторы функции f1

```
}
```

```
int f2() {
```

операторы функции f2

```
}
```

Простейший ввод-вывод

```
#include <iostream>
```

Вывод:

```
cout << "x=" << x << endl;
```

Ввод:

```
cin >> x;
```

Простейший ввод-вывод в стиле C

```
#include <stdio>
```

Вывод:

```
printf ("x=%d", x) ;
```

Ввод:

```
scanf (&x) ;
```

Константы

Вид	Примеры
<i>Целые</i> Дес.	8 0 199226
Восьм.	01 020 07155
<u>Шестн.</u>	<u>0xA</u> <u>0x1B8</u> <u>0X00FF</u>
<i>Веществ.</i>	5.7 .001 35.
<u>Вещ. с плав. т.</u>	<u>0.2E6</u> <u>.11e-3</u> <u>5E10</u>
<i>Символьные</i>	'A' 'ю' '*' 'd' '\0' <u>'\n'</u> <u>'\012'</u> <u>'\x07\x07'</u>
<i>Строковые</i>	"Здесь был Vasia" " \tЗначение r=\0xF5\n"

Типы данных

Тип данных определяет:

- *внутреннее представление* данных в памяти компьютера => *множество значений*, которые могут принимать величины этого типа;
- *операции и функции*, которые можно применять к величинам этого типа.

Типы в C++ делятся на *основные* (fundamental) и *составные* (compound). Тип может описывать объект, ссылку или функцию.

Основные (стандартные) типы данных:

int (целый);

integer

char (символьный);

wchar_t (расширенный символьный);

bool (логический);

float (вещественный);

double (вещественный с двойной точностью).

Спецификаторы:

short (короткий);

long (длинный);

signed (знаковый);

unsigned (беззнаковый).

+ void

Диапазоны для IBM PC-совместимых

Тип	Диапазон значений	Размер(байт)
<code>bool</code>	<code>true</code> и <code>false</code>	1
<code>signed char</code>	-128 ... 127	1
<code>unsigned char</code>	0 ... 255	1
<code>signed short int</code>	-32 768 ... 32 767	2
<code>unsigned short int</code>	0 ... 65 535	2
<code>signed long int</code>	-2 147 483 648 ... 2 147 483 647	4
<code>unsigned long int</code>	0 ... 4 294 967 295	4
<code>float</code>	$3.4e-38$... $3.4e+38$	4
<code>double</code>	$1.7e-308$... $1.7e+308$	8
<code>long double</code>	$3.4e-4932$... $3.4e+4932$	10

Вещественные числа

Одинарная точность

1 бит 8 бит 23 бита

Зн	Порядок	Мантисса
----	---------	----------

Двойная точность

1 бит 11 бит 52 бита

Зн	Порядок	Мантисса
----	---------	----------

Расширенная точность

1 бит 15 бит 64 бита

Зн	Порядок	Мантисса
----	---------	----------

Диапазоны типов по стандарту

- $\text{sizeof}(\text{float}) \leq \text{sizeof}(\text{double}) \leq \text{sizeof}(\text{long double})$
- $\text{sizeof}(\text{char}) \leq \text{sizeof}(\text{short}) \leq \text{sizeof}(\text{int}) \leq \text{sizeof}(\text{long})$

Описание идентификаторов

[класс памяти] [const] тип имя [инициализатор];

инициализатор: = значение ИЛИ (значение)

Примеры описаний:

```
short int a = 1;
```

```
const char C = 'C';
```

```
char s, sf = 'f';
```

```
char t (54);
```

```
float c = 0.22, x(3), sum;
```

Класс памяти

auto — *автоматическая* переменная. Память выделяется в стеке и при необходимости инициализируется каждый раз при выполнении оператора, содержащего ее определение. Освобождение памяти - при выходе из блока

extern — переменная определяется в другом месте программы.

static — *статическая* переменная. Время жизни — постоянное. Инициализируется один раз при первом выполнении оператора, содержащего определение переменной. В зависимости от расположения оператора описания статические переменные могут быть *глобальными и локальными*.

register — аналогично auto, но память выделяется по возможности в регистрах процессора.

Область видимости

```
int a;           // глобальная переменная

int main(){

    int b;       // локальная переменная

    static int c = 1; // локальная статическая переменная

}
```

	Глобальная	Локальная	Статическая
Размещение	с-т данных	с-т стека	с-т данных
Время жизни	вся прогр.	блок	вся прогр.
Область видимости	файл	блок	блок
Обнуление	да	нет	да

Область видимости

- блок
- файл
- функция
- прототип функции
- класс
- поименованная область

Пример программы 1

```
#include <stdio.h> //Подключение библиотеки
int main() //Главная функция программы
{ //начало функции(блока)
    int i;
    printf(“Hello world!!!”); //Вывод текста на
        //экран
    return 0; //Завершение работы(возврат из
        //функции)
} //Конец функции(блока)
```

Пример программы 2

```
#include <iostream> //Подключение  
    //библиотеки  
using namespace std;  
int main() //Главная функция программы  
{//начало функции(блока)  
    int i;  
    cout<<“Hello world!!!”<<endl; //Вывод текста  
        //на экран  
    return 0; //Завершение работы(возврат из  
        //функции)  
} //Конец функции(блока)
```

Операции в C++

Унарные операции:

-	Изменение знака
++	Инкремент
--	Декремент
(type)	Преобразование к типу
sizeof()	Определение размера в байтах
&	Определение адреса
*	Разадресация

Бинарные арифметические операции:

-	Вычитание
+	Сложение
*	Умножение
/	Деление
%	Остаток от деления
>>	Сдвиг вправо
<<	Сдвиг влево

Поразрядные логические операции:

\sim	Поразрядное логическое НЕ
$\&$	Поразрядное логическое И
$ $	Поразрядное логическое ИЛИ
\wedge	Поразрядное исключающее ИЛИ

Логические операции:

!	Логическое отрицание
&&	Логическое И
	Логическое ИЛИ

Операции отношения:

$>$	Больше
$<$	Меньше
\geq	Больше или равно
\leq	Меньше или равно
$==$	Равно
\neq	Не равно

Базовые конструкции

- **Следованием** называется конструкция, представляющая собой последовательное выполнение двух или более операторов (простых или составных).
- **Ветвление** задает выполнение либо одного, либо другого оператора в зависимости от выполнения какого-либо условия.
- **Цикл** задает многократное выполнение оператора.

Базовые конструкции

- Особенностью базовых конструкций является то, что любая из них имеет только один вход и один выход, поэтому конструкции могут вкладываться друг в друга произвольным образом, например, цикл может содержать следование из двух ветвлений, каждое из которых включает вложенные циклы

Оператор if

```
if (expression) TRUE_statement  
[else FALSE_statement]
```

Statement – простой или составной оператор

Составной оператор – группа операторов
заклученная в фигурные скобки { }

Примеры:

```
if (a<0) b = 1;
```

```
if (a<b && (a>d || a==0)) b++;  
else {b* = a; a = 0;}
```

```
if (a<b) {if (a<c) m = a; else m = c;}  
else {if (b<c) m = b; else m = c;}
```

```
if (a++) b++;
```

```
if (b>a) max = b; else max = a;
```


Важно!

- Распространенная ошибка при записи условных операторов — *использование в выражениях вместо проверки на равенство (`= =`) простого присваивания (`=`),*
- например, `if(a=1)b=0;.`

Оператор ветвления switch

```
switch (expression)
{
    case constant1: statement1; [break;]
    ...
    case constant_i: statement_i; [break;]
    ...
    case constant_N: statement_N; [break;]
    [default: statement(N+1);]
}
```

Выход из оператора ветвления

- Выход из переключателя обычно выполняется с помощью операторов **break**.
- Оператор **break** выполняет выход из самого внутреннего из объемлющих его операторов **switch**, **for**, **while** и **do**.
- Оператор **return** выполняет выход из функции, в теле которой он записан.

Пример оператора ветвления

```
switch (op){  
    case '+': res = a + b; break;  
    case '-': res = a - b; break;  
    case '*': res = a * b; break;  
    case '/': res = a / b; break;  
    default : cout << "\nНеизвестная  
операция";  
}
```

Заключение

- Спасибо за внимание!
- Вопросы???