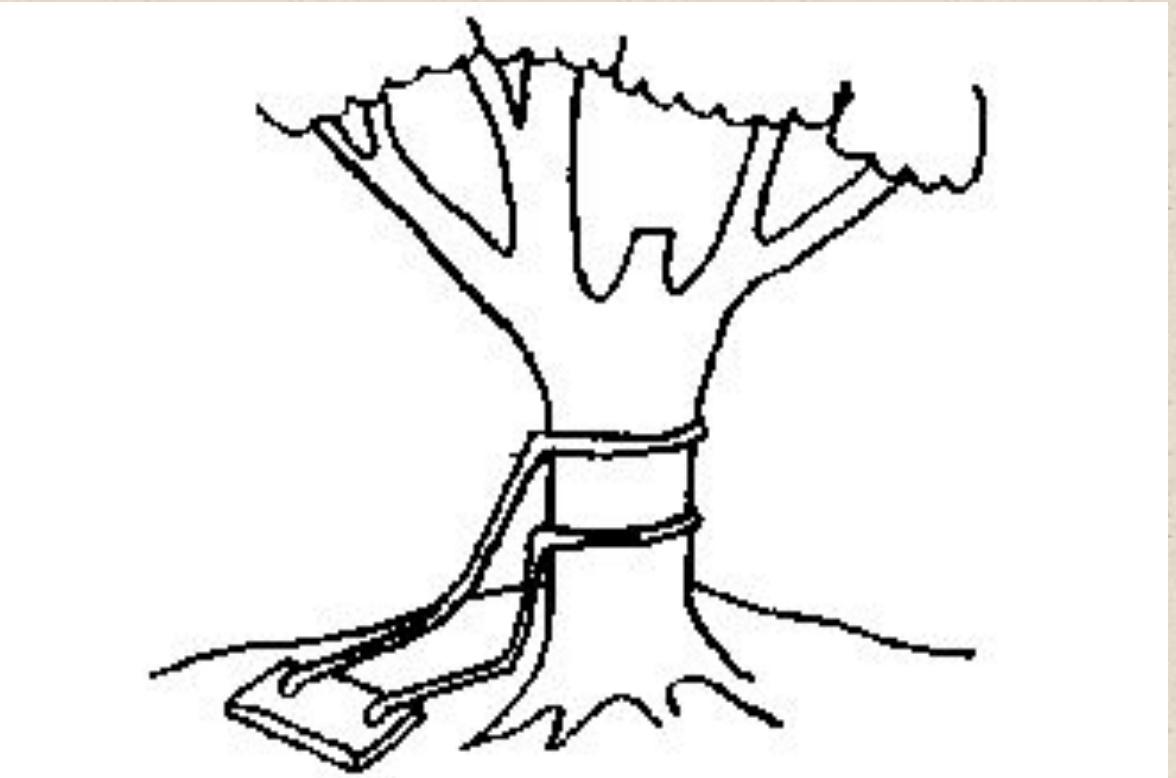
Вычислительная техника и компьютерное моделирование в физике

Зинчик Александр Адольфович
zinchik_alex@mail.ru

«Качели» - как проектируются программы :)

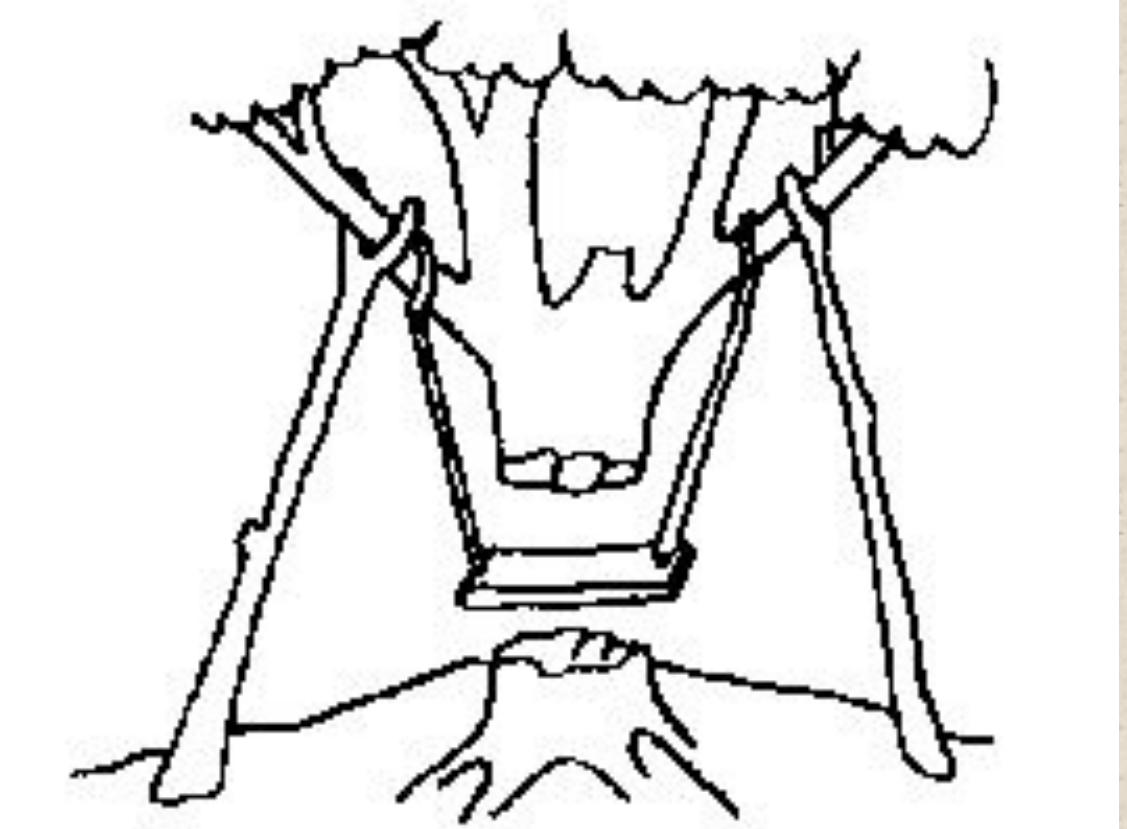
Как было предложено
организатором разработки

Как было описано
в техническом задании



Как было спроектировано
ведущим системным специалистом

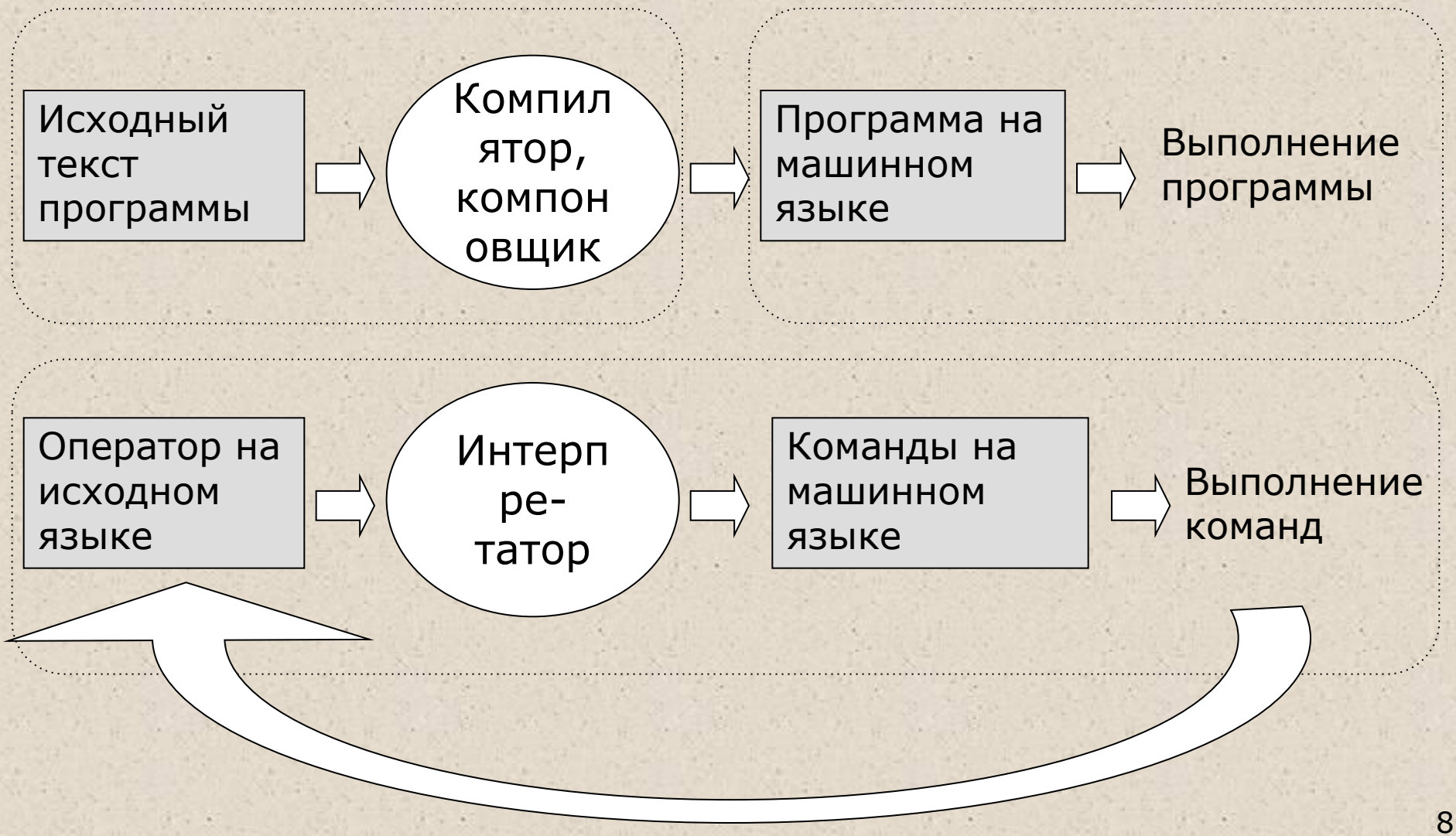
Как было реализовано программистами



Как было внедрено

Чего хотел пользователь

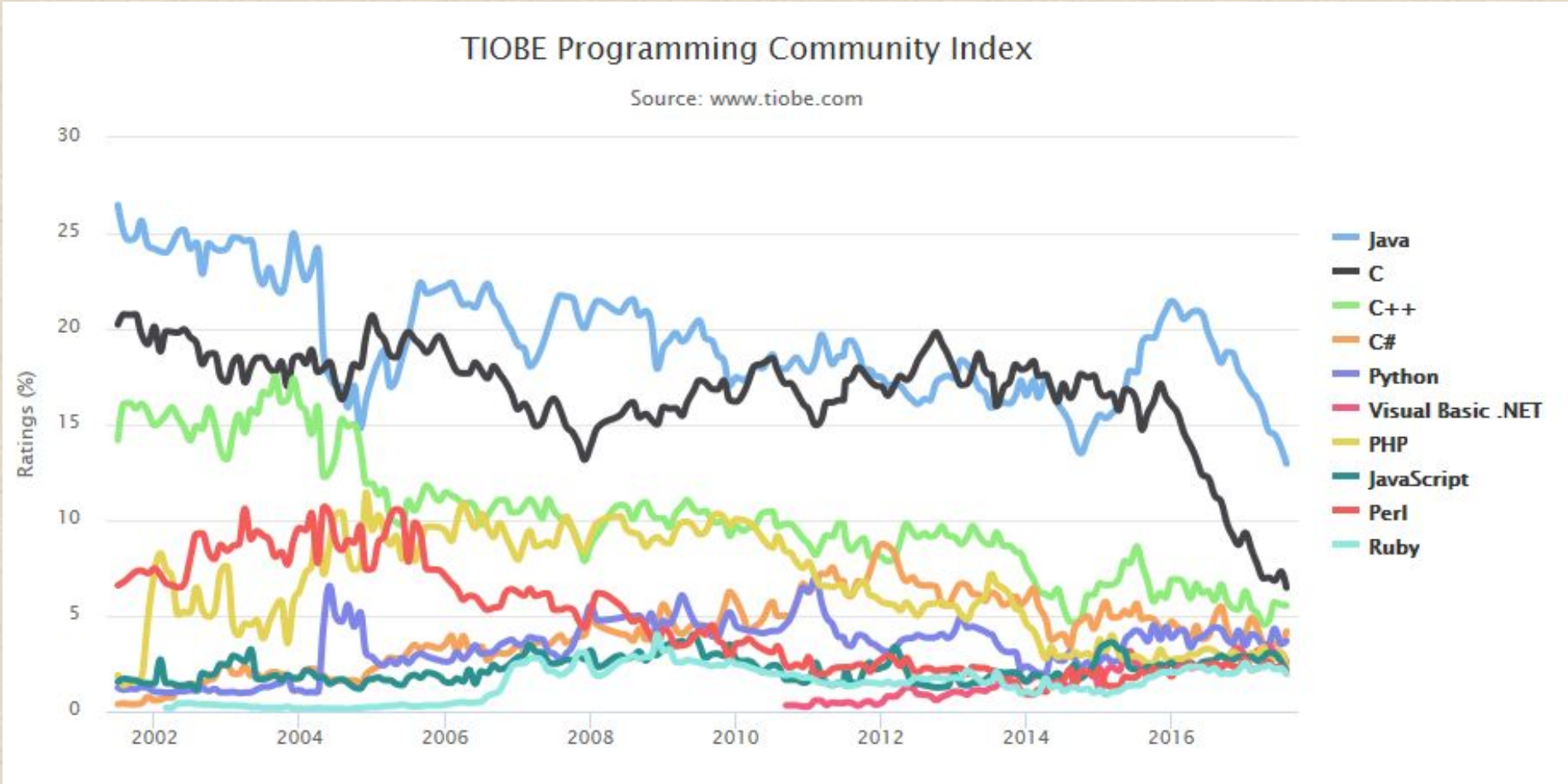
Компилятор или интерпретатор



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



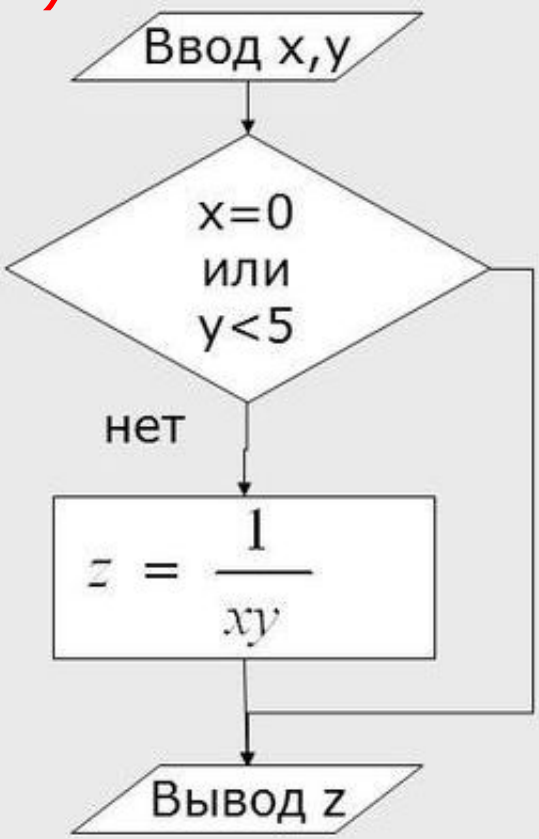
Небольшой тест

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.

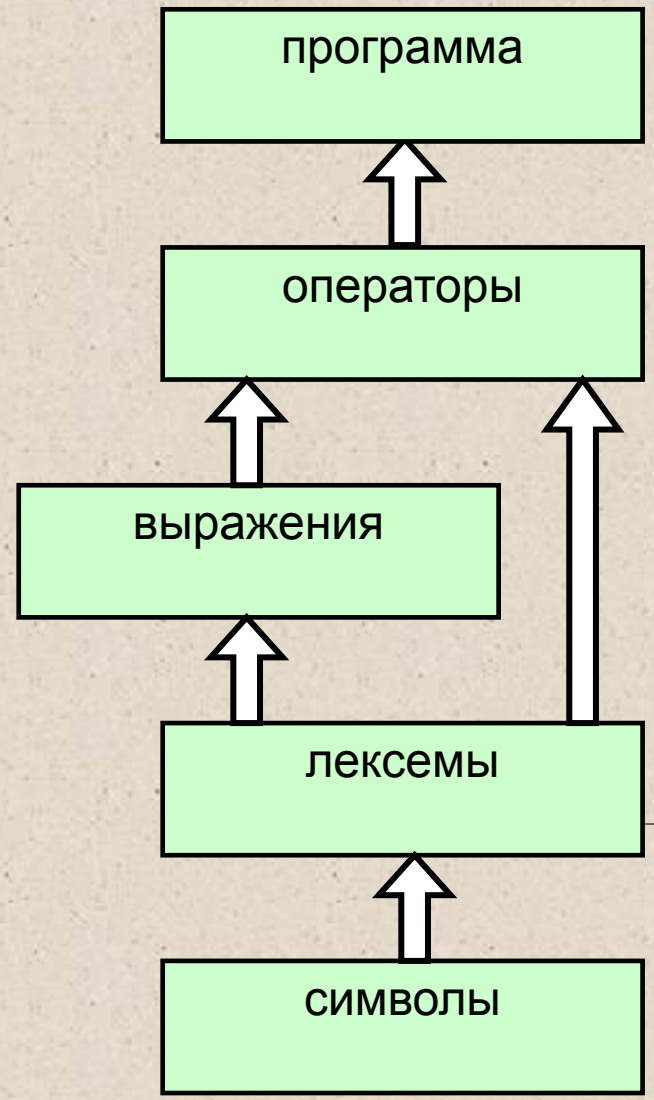
- *Процедурная декомпозиция* состоит в том, что задача, реализуемая программой, делится на подзадачи, а они, в свою очередь — на более мелкие этапы, то есть выполняется пошаговая детализация алгоритма решения задачи.

- *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



примеры

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, “, {,},|,/,%,…

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 <cstdio>
Вывод:
printf(“x=%d“,x);
Ввод:
scanf(&x);
```

Константы

Вид	**Примеры**
Целые Дес.	8 0 199226
Восьм.	01 020 07155
Шестн.	0xA 0x1B8 0X00FF
Веществ.	5.7 .001 35.
Вещ. с плав. т.	0.2E6 .11e–3 5E10
Символьные	'A‘ 'ю‘ '*‘ ‘d‘ '\0' '\n‘ '\012‘ '\x07\x07'
Строковые	"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).

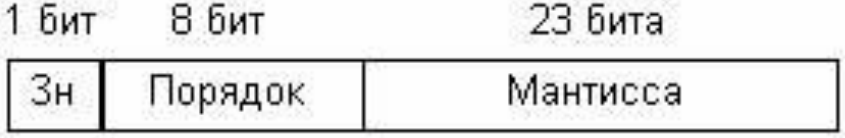
+ void

Диапазоны для IBM PC-совместимых

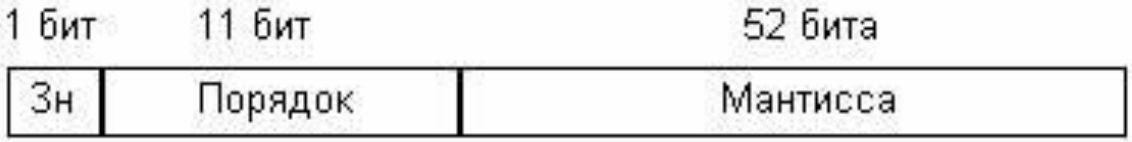
Тип	Диапазон значений	Размер(байт)
bool	true и false	1
signed char	-128 … 127	1
unsigned char	0 … 255	1
signed short int	-32 768 … 32 767	2
unsigned short *int*	0 … 65 535	2
signed long int	-2 147 483 648 … 2 147 483 647	4
unsigned long int	0 … 4 294 967 295	4
float	3.4e-38 … 3.4e+38	4
double	1.7e-308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

Вещественные числа

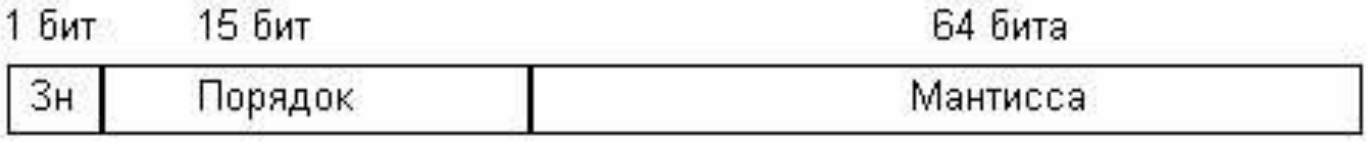
Одинарная точность



Двойная точность



Расширенная точность



Диапазоны типов по стандарту

- sizeof(float) ≤ sizeof(double) ≤ sizeof(long double)
- sizeof(char) ≤ sizeof(short) ≤ sizeof(int) ≤ sizeof(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:

~	Поразрядное логическое НЕ
&	Поразрядное логическое И
|	Поразрядное логическое ИЛИ
^	Поразрядное исключающее ИЛИ

Логические операции:

!	Логическое отрицание
&&	Логическое И
||	Логическое ИЛИ

Операции отношения:

>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно
==	Равно
!=	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 заключенная 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???