Введение в язык программирования C.

Лекция 1

Сергей Леонидович Бабичев

Что такое алгоритм и его исполнители.

Алгоритм и его исполнитель

Алгоритм — это последовательность команд для *исполнителя*, обладающая рядом свойств:

- **полезность**, то есть умение решать поставленную задачу.
- **детерминированность**, то есть каждый шаг алгоритма должен быть строго определён во всех возможных ситуациях.
- **конечность**, то есть способность алгоритма завершиться для любого множества входных данных
- **массовость**, то есть применимость алгоритма к разнообразным входным данным.

Алгоритм *исполняется* и требует *ресурсов*.
Программа есть запись алгоритма на формальном языке.

Цель алгоритма

- Обработка данных: *входные данные* $\xrightarrow{\text{алгоритм}}$ *выходные данных*.
- При исполнении могут появиться *промежуточные данные*.
- Удобно, когда данные *упорядочены*, образуя *структуры данных*.

Исполнение алгоритма

- Одна задача — несколько алгоритмов — разные используемые ресурсы.
- Разные исполнители алгоритмов.
- Элементарные действия и элементарные объекты.
- Исполнитель *Центральный процессор* — очень мелких операций над ограниченным набором данных.
- Исполнитель *Python* — операции над *объектами*, использует *центральный процессор*.

Языки программирования

- Нужно записать алгоритм так, чтобы не было сомнений, как его исполнять.
- Язык программирования — способ *формализовать* запись алгоритма.
- Программа на языке программирования — входные данные для *транслятора* с этого языка.

Виды трансляции

- *Интерпретация* — обработка входной программы шаг за шагом. Python.

  prog.py $\xrightarrow[\text{Интерпретатор Python}]{\text{Входные данные программы}}$ результат.

- *Компиляция* — перевод всей программы сразу на язык другого исполнителя. C. C++.

  prog.c $\xrightarrow{\text{Компилятор C}}$ prog.exe

  Входные данные программы $\xrightarrow{\text{prog.exe}}$ результат.

Сравнение интерпретации и компиляции

- Интерпретация:
  - Нет промежуточных этапов.
  - Интерактивность.
  - Гибкость.
- Компиляция:
  - Для исполнения не нужны дополнительные компоненты.
  - Высокая скорость исполнения алгоритма.

Программа и её исполнение

- Программа — нечто, реализующее алгоритм.
- Имеются входные данные.
- Имеются выходные данные.
- В первом приближении язык, на котором написана программа, можно рассматривать как исполнителя этой программы.
- Нужно рассмотреть с чем умеет работать исполнитель (*типы данных*) и что он умеет делать (*операции*).

Язык программирования C

Чем хорош язык C

- Синтаксис этого языка послужил основой для синтаксиса таких языков, как `C++`, `Java`, `C#`. Изучив `C`, достаточно легко переключиться на более новые языки.
- Близок к машине, программы написанный на нём исполняются быстро.
- Очень компактен. Для его изучения не требуется много времени.
- Много готового кода уже написано и он имеется только на `C`

Типы данных C

- Можно создавать сложные объекты из простых.
- Самые простейшие объекты делятся на две группы:
  - Целочисленные
    - Количество битов.
    - Наличие или отсутствие знака.
  - Вещественные

Целочисленные типы

Не забудьте, в C нет *длинных* чисел! Это — не `Python`!

- Сколько разных чисел можно закодировать 8 битами?
- Какие это могут быть числа?
- Если числа без знака, то диапазон $[0..255]$.
- Если числа со знаком, то диапазон $[-128..127]$.

Объявление целочисленных типов

Всё, что мы обрабатываем в C, должно быть где-то *объявлено*. Объявление выделяет место под хранение значений и устанавливает *тип данных*.

Несколько вариантов:

- `char` — обычно знаковый тип, 8 битов. $(-128...127)$.
- `signed char` — точно знаковый тип. $(-128...127)$.
- `unsigned char` — беззнаковый тип, 8 битов. $(0..255)$.
- `short` — знаковый тип, обычно 16 бит. $(-32768..32767) = (-2^{15}..2^{15}-1)$.
- `unsigned short` — беззнаковый тип, обычно 16 бит. $(0..65535) = (0..2^{16}-1)$.
- `int` — знаковый тип для «естественных» для данной вычислительной системы целых чисел. Обычно 32 бита, $(-2^{31}..2^{31}-1)$ или $(-2\ 147\ 483\ 648..2\ 147\ 483\ 647)$.
- `unsigned int` — то же, но беззнаковое. $(0..2^{32}-1)$ или $(0..4\ 294\ 967\ 295)$.
- `long long int` — знаковый тип, 64 бита, $(-2^{63}..2^{63}-1)$.
- `unsigned long long int` — беззнаковый тип, 64 бита, $(0..2^{64}-1)$

Вещественные числа

- Любой отрезок прямой содержит бесконечное количество вещественных чисел и конечное целых.
- В компьютере невозможно представить все возможные вещественные числа даже на отрезке. Берётся некоторое подмножество, которое можно закодировать конечным числом бит. Ряд чисел при этом может быть представлен точно, все остальные — приближённо.
- Важны два параметра: диапазон представления чисел и количество значащих цифр в числе. Всё кодируется в двоичном представлении.

Различие между целыми и действительными числами

Фундаментальное различие между целыми и вещественными числами на компьютере следующее:

- Целые числа представляют информацию точно, но *дискретность* представления равна единице, поэтому *абсолютная погрешность* представления произвольного числа (при условии попадания числа в диапазон) постоянна.
- Вещественные числа представляют информацию *приближённо*, диапазон представления больше, чем у целых чисел и *относительная погрешность* постоянна.

Виды вещественных чисел

- `float` — ещё называемые *одиночной точности*. Диапазон представления от примерно $-10^{38} \dots 10^{38}$, количество значащих цифр в представлении от 6 до 7, зависит от числа.
- `double` — числа *двойной точности*. Диапазон примерно от $-10^{308} \dots 10^{308}$. Количество значащих цифр — $16 - 17$.
- На ряде вычислительных систем `long double` или `extended`, с ещё более широким диапазоном.

Составные типы

Любые типы можно объединять друг с другом, образуя:

- *массивы* — несколько объектов одного типа, располагающихся рядом друг с другом и требующих *индекса* для выбора того элемента, который мы имеем в виду.
- *структуры* (`struct`) — несколько *полей* (*fields*) произвольных типов, требующих *селектора* для идентификации требуемого. Например, мы можем создать структуру `complex`, в которой будут поля `re` и `im`.
- *объединения* `union` — несколько полей разделяют один *адрес*, то есть хранятся в одном и том же месте.

Каждое поле структуры или элемент массива могут быть, в свою очередь, структурой (объединением) или массивом.

Литералы

Литерал — элемент записи программы, который представляет сам себя.

Числа являются литералами. Каждый литерал имеет тип. Литералы могут иметь *префиксы* и *суффиксы*.

Комментарий — часть текста программы, не влияющая на её исполнение.

```
// однострочный комментарий.
/* много
строчный
комментарий
*/
```

Примеры литералов с суффиксами

```
1         // тип int
1l        // буква l - суффикс типа long.
1L        // буква L, и большая и малая - суффикс типа long
1U         // буквы U - суффикс беззнакового типа. unsigned
123UL      // и long, и unsigned
11111ULL  // unsigned long long
123f       // float
123d       // double
123.0      // double
123E17     //  double, суффикс "экспонента", 123 * 10^17
6.02E23    // double, число Авогадро в СГС
'A'        // char, значение равно коду символа A, то есть 65
```

Примеры литералов с префиксами

```
 0x123   // 0x - шестнадцатеричное представление числа
 0666    // 0 - восьмеричное представление
 '\x12'  // Константа char с кодом 12 в шестнадцатеричной
системе
 '\0'    // Константа char с кодом 0
```

Строчные литералы

Строчный литерал начинается с *двойной кавычки*, ей же заканчивается:

```
"This is a string literal"
```

Это — пример литерала-*массива*.

> Не путайте одиночные и двойные кавычки, `'0'`— число, `"0"` — массив.

Символ \, *бэкслэш*, в символьных и строчных литералах играет специальную роль — он изменяет значение следующего за ним символа. Вот некоторые константы типа `char`, которые не имеют печатного представления:

```
'\n'    // Переход на новую строку
'\r'    // Переход на начало строки
'\t'    // Знак табуляции
'\b'    // Возврат на 1 шаг назад
'\\'    // Сам бэкслэш
"Hello\n"  // Слово Hello и переход на новую строку
```

Переменные

```
int a,b,c; // Три переменных целого типа с именами a,b,c
char qwerty; // Переменная типа char с именем qwerty
double d;   // Переменная типа double с именем d
```

Мы *объявили* переменные. Выделили под них память. Значения переменных не определены.
Лучше делать так (*инициализировать* переменные):

```
int n = 100, m = 500;
double pi = 3.14159265356793;
char delim = '\n';
unsigned mask = 0xFFFF;
```

Идентификатор

Идентификаторы именуют объекты языка. Только буквы латинского алфавита в любом регистре, знак подчёркивания _ и цифры. Цифра не может быть первым символом идентификатора.

- Корректные идентификаторы: `abracadabra`, `N`, `pn766576`, `_`, `__FILENAME__`.
- Некорректные: `Привет`, `1p`.

Прописные и строчные буквы в идентификаторы различаются (это не `Pascal`). Имена `n` и `N` различны.

Ключевые слова

Некоторые идентификаторы — *зарезервированные* идентификаторы или *ключевые слова*. Их 34 штуки.

`auto`, `break`, `case`, `char`, `const`, `continue`, `default`, `do`, `double`, `else`, `enum`, `extern`, `float`, `for`, `goto`, `if`, `inline`, `int`, `long`, `register`, `restrict`, `return`, `short`, `signed`, `sizeof`, `static`, `struct`, `switch`, `typedef`, `union`, `unsigned`, `void`, `volatile`, `while`.

Не используйте ключевые слова для имён переменных и функций.