# ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ
# ДЕКОМПОЗИЦИЯ. ДВИЖЕНИЕ ПО ЛАБИРИНТУ

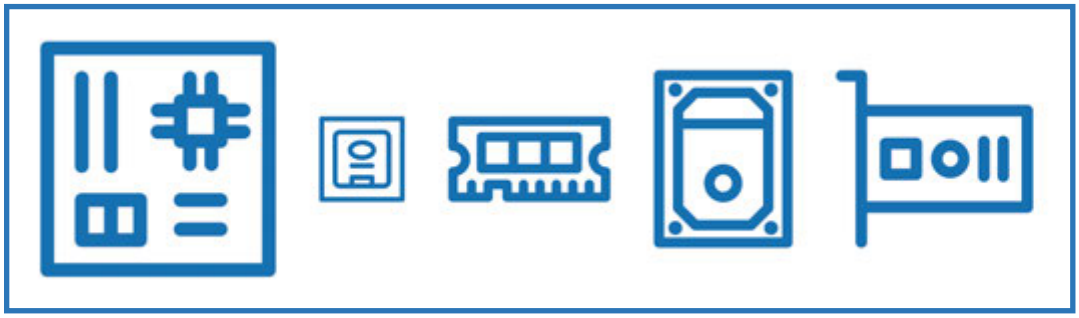
Ранее нами уже было рассмотрено понятие «система». Вспомним, что под системой понимают набор взаимосвязанных между собой элементов, которые составляют единое целое в рамках рассматриваемой задачи. Однако, говоря о системе, нельзя не упомянуть и связанные с ней подсистему и надсистему. На этих понятиях строится одна из важнейших технологий современности – стандартизация.

Для того чтобы представить сущность и взаимосвязь этих понятий, давайте разберем устройство компьютера.

Если посмотреть на персональный компьютер, то мы увидим, что он состоит из:

- системного блока, в котором находится все аппаратное обеспечение компьютера (электронные части, которые производят вычисления);
- клавиатуры;
- мыши;
- монитора;
- устройства воспроизведения звука.

Каждый из названных элементов компьютера состоит в свою очередь из других элементов. Например, системный блок состоит из:



- материнской платы, на которой соединяются воедино все устройства компьютера;
- процессора - устройства, которое производит вычисления и управляет работой всего компьютера в целом;
- оперативной памяти - устройства, в котором во время работы компьютера хранятся выполняемый машинный код (программы) и данные, обрабатываемые процессором (при выключении компьютера данные из оперативной памяти исчезают);
- постоянной памяти – устройства, в котором хранятся данные и после выключения компьютера; обращение к этим данным занимает больше времени, чем обращение к данным в оперативной памяти;
- видеокарты – по сути, компьютера внутри компьютера, предназначенного прежде всего для быстрой обработки видеоинформации.

Обратите внимание, что любое устройство, принадлежащее системе компьютер, само является системой. В таком случае удобно объединять несколько устройств с одинаковыми свойствами в подсистемы.

Таким образом, в системе персонального компьютера есть огромное количество подсистем, а сам он может являться элементом огромного количества надсистем, например частью компьютерной сети интернет.

Значит ли это, что все элементы системы компьютер должны быть произведены одним и тем же производителем?

Если бы это предположение было верным, то на рынке существовало бы множество различных компьютеров: у одних были бы лучше одни элементы, у других -другие. Но самая большая сложность состояла бы в том, что программные продукты,

созданные для одних компьютеров, не работали бы на других. На самом деле при зарождении рынка персональных компьютеров именно так и происходило. Из этой

ситуации был найден следующий выход. Во-первых, были разработаны стандарты соединения и обмена данных элементов системы между собой. Это привело к тому, что пользователь получил возможность выбрать одного производителя материнской платы и совсем другого производителя оперативной памяти. Во-вторых, были разделены производители программного и аппаратного обеспечения. Для того чтобы все это реализовать, необходимо было разработать образцы (эталон, стандарт), которые обеспечили бы совместимость. Например, для связи устройств ввода сегодня чаще всего используется разъем USB, для которого обеспечен свой стандарт передачи данных, размера гнезда:


Что такое стандартизация?

**Стандартизация - это деятельность по разработке, опубликованию и применению стандартов, по установлению норм, правил и характеристик в целях обеспечения безопасности продукции, работ и услуг, технической и информационной совместимости, взаимозаменяемости и качества продукции, работ и услуг.**

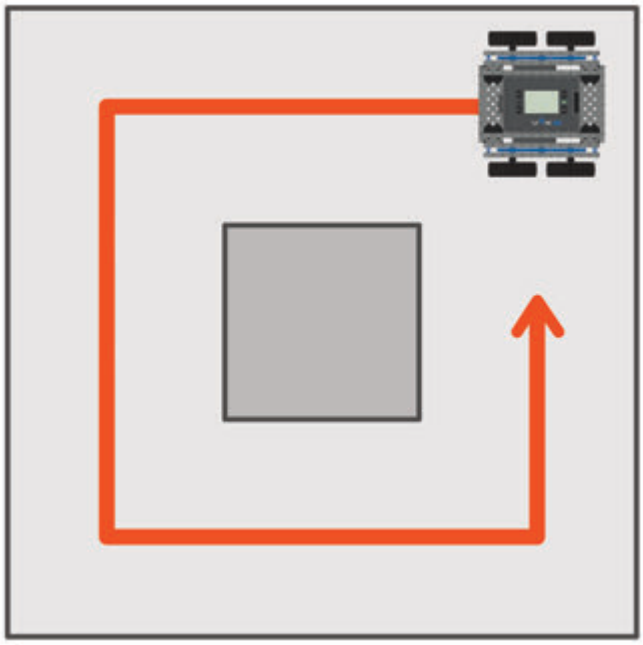
Подобно персональным компьютерам стандартизация проникла почти во все сферы производства продукции и услуг. Так, например, производители самолетов не производят кресла и двигатели, а производители автомобилей - шины и автомобильные дворники. Для автомобилей также разработаны стандарты топлива, чтобы разные авто могли использовать одно и то же топливо. В основе стандартизации лежит так называемая декомпозиция.

Что такое декомпозиция?

**Декомпозиция - это разделение целого на части. Также декомпозиция - это научный метод, использующий структуру задачи и позволяющий заменить решение одной большой задачи решением серии меньших задач, пусть и взаимосвязанных, но более простых.**

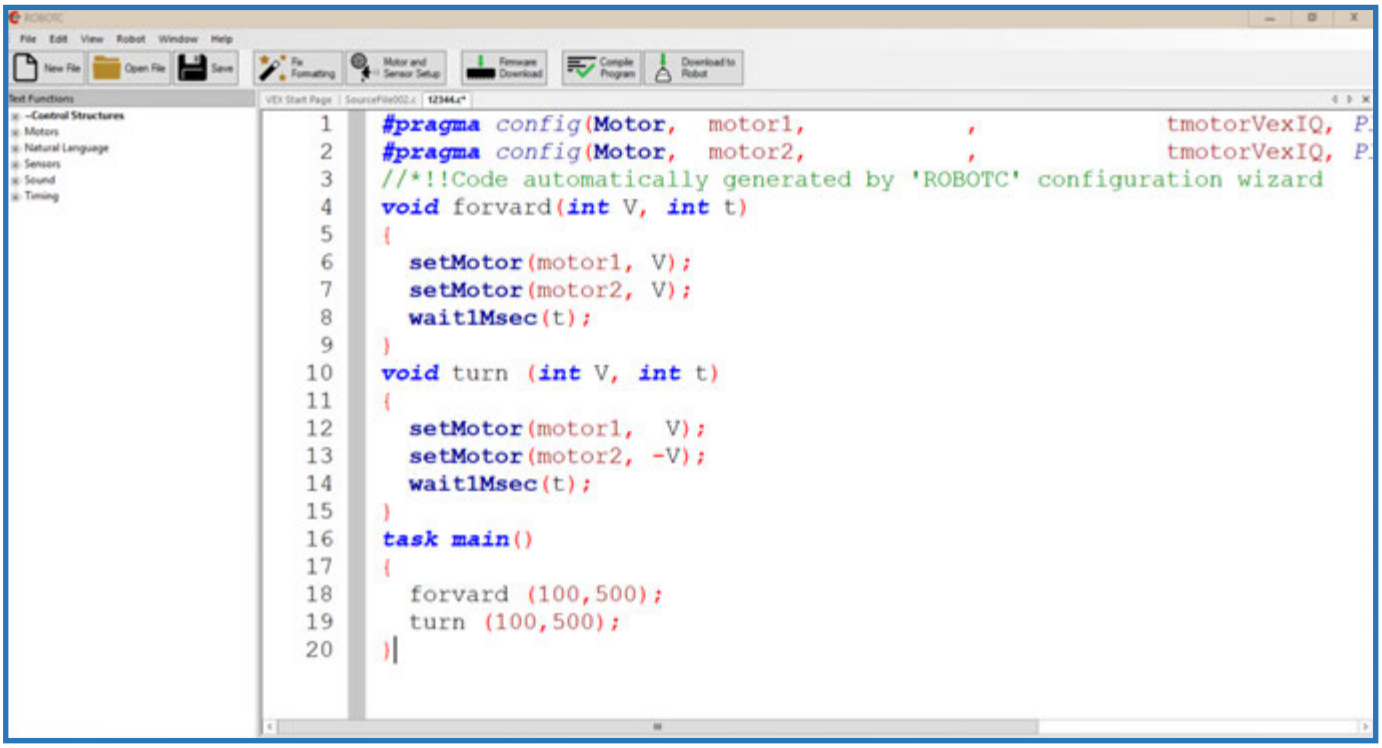
Выше нами уже была рассмотрена декомпозиция персонального компьютера, давайте попробуем применить ее и для программного кода, с помощью которого осуществляется управление роботом, например при движении по лабиринту.

Что такое лабиринт? Это какая-либо структура, состоящая из множества прямых отрезков пути, соединенных между собой под углом 90 градусов. У него могут быть стены или он может быть нарисован (напечатан). В любом случае траекторию движения робота можно разбить на три повторяющиеся составляющие: движение прямо, разворот по часовой стрелке и разворот против часовой стрелки на 90 градусов. В самом простом своем варианте лабиринт состоит из серии квадратов. Для наглядности возьмем замкнутый лабиринт и представим себе код для движения робота по нему без использования сенсоров.



Роботу будет необходимо пройти четыре стороны квадрата и при этом 3 раза повернуть на 90 градусов. Это достаточно большое количество строк кода. Для того чтобы сделать код более понятным и удобным, воспользуемся новой для нас технологией – декомпозицией.

При декомпозиции в программировании следует найти самые важные и наименьшие части и представить их в виде функций. Например, задачу для движения по прямоугольнику можно разбить на две повторяющиеся функции: движение прямо и поворот на 90 градусов.

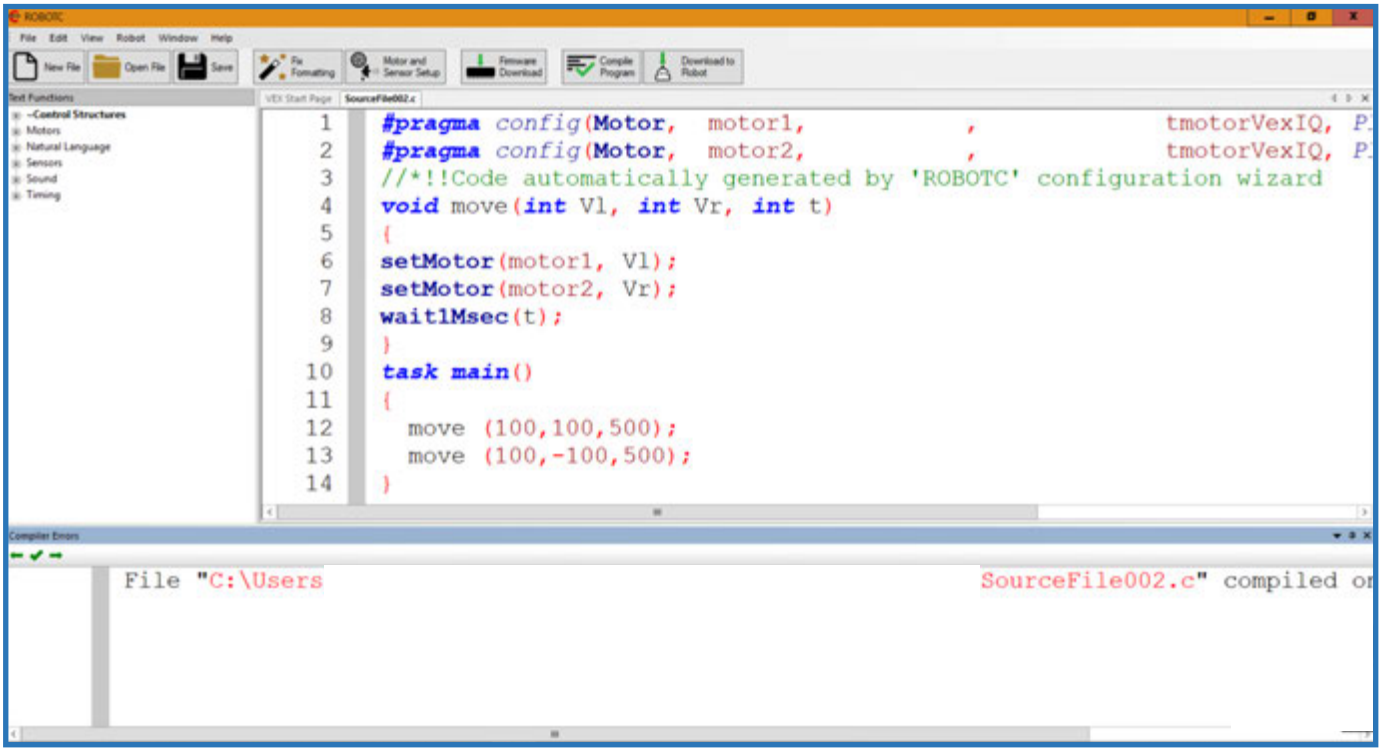


```
1  #pragma config(Motor,  motor1,           ,             tmotorVexIQ, P
2  #pragma config(Motor,  motor2,           ,             tmotorVexIQ, P
3  //*!!Code automatically generated by 'ROBOTC' configuration wizard
4  void forvard(int V, int t)
5  {
6    setMotor(motor1, V);
7    setMotor(motor2, V);
8    wait1Msec(t);
9  }
10 void turn (int V, int t)
11 {
12   setMotor(motor1,  V);
13   setMotor(motor2, -V);
14   wait1Msec(t);
15 }
16 task main()
17 {
18   forvard (100,500);
19   turn (100,500);
20 }
```

Нами было создано две функции типа **void**. Особенность этой функции заключается в том, что она не возвращает никакого значения после вызова. И пока не будет выполнена она, не будет выполнено и никакое другое действие в программе. В нашем случае, пока в 18 строке не будет закончено выполнение функции **forvard**, не начнется и выполнение функции **turn**.

Функции **forvard** и **turn** имеют одинаковые аргументы, которые записаны внутри круглых скобок сразу после названия функции: это целочисленные переменные V и t. Первая отвечает за скорость вращения двигателей, вторая - за время движения. Тела же функций, заключенные между фигурных скобок, отличаются только знаком для скорости второго мотора. Таким образом, после вызова функции **forvard** с ненулевым значением скорости робот будет перемещаться вперед или назад, при вызове же функции **turn** - разворачиваться влево-вправо.

Такая декомпозиция является вполне приемлемой для двух типов движений: поступательного и разворота вокруг центра робота - но она не годится для реализации движения робота по дуге. Следующая декомпозиция позволит реализовать все возможные варианты движения робота:




В функции **move** три аргумента: скорость левой пары колес, скорость правой пары колес, время продолжения работы двигателя. Эта декомпозиция является наиболее универсальной и именно ее мы и будем использовать в дальнейшем.

Итак, для того чтобы написанный для управления роботом код не был громоздким и неудобочитаемым, используется декомпозиция. При этом в поставленной перед роботом задаче выделяются более мелкие значимые задачи, которые и представляются в виде функций, выполняющихся последовательно или параллельно.