Указатели. Проще простого

Практический курс по C/C++: https://stepik.org/course/193691

Смотреть материал на YouTube | RuTube

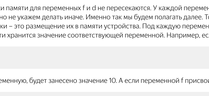
Прежде чем двигаться дальше, изучать массивы, строки, функции и т.п., нужно вначале познакомиться с самым прекрасным, сакраментальным и основополагающим знанием – концепцией указателей. Это невероятно простая тема, вводящая многих в трепетный ужас. С чем это связано? Не знаю. Для меня это загадка. Чтобы преодолеть «проклятие» понимания указателей, я, на этом занятии, рассажу с них предельно подробно, красочно и ёмко. А начнем, как всегда, с самого начала.

Из предыдущих серий вы уже знаете, что вычислительная техника снабжена оперативной памятью, которую можно представить в виде непрерывной последовательности ячеек, размером в один байт. Каждая ячейка имеет свой номер, который называется **адресом**.

Также вы знаете, что в ячейки располагаются программы и данные. Но нас будут интересовать только данные, которые на уровне языка описываются переменными и константами. Поэтому предположим, что в программе объявляется переменная типа char:

```
char d;
```

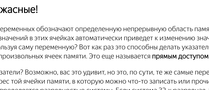
В результате, когда выполнение дойдет до этой строчки, в памяти устройства автоматически будет выделена область памяти размером в один байт (одну ячейку):



Если затем в программе объявляется еще одна переменная, предположим, типа int:

```
int f;
```

то также автоматически где-то в памяти будет выделена свободная область для хранения этой переменной. Причем, под нее будет уже отведено четыре подряд идущих байта. При этом адресом переменной считается номер первой ячейки. В нашем примере **адрес переменной f равен 101**, несмотря на то, что она оказывает еще три ячейки с адресами 102, 103 и 104.



Также обратите внимание, что ячейки памяти для переменных f и d не пересекаются. У каждой переменной строго своя область. И это наиболее частый вариант, если сами, намеренно не указем делать иначе. Именно так мы будем полагать далее. То есть, смысл объявления переменных с точки зрения вычислительной техники – это размещение их в памяти устройства. Под каждую переменную где-то в памяти отводится свои независимая область и в этой области хранится значение соответствующей переменной. Например, если переменной d присвоить значение 10:

```
d = 10;
```

то в ячейке, отведенной под эту переменную, будет занесено значение 10. А если переменной f присвоить значение 75432:

```
f = 75432;
```

то в ее ячейках будут записаны числа: 168, 38, 1, 0. Почему именно такие? Потому, что число 75432 кодируется на уровне байт следующим образом:

75432 = 168 + 256 · 38 + 256^2 · 1

И аналогичная картина будет для любых переменных языка Си. Например, при объявлении вещественной переменной t типа double:

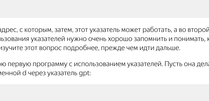
```
double t;
```

в памяти будет выделено уже 8 байт. И то, что содержится в этих 8 байтах и будет определять содержимое этой переменной. Так хранятся любые переменные, которые мы объявляем в программах.

Указатели, страшные и ужасные!

Практически полчается, что имена переменных обозначают определенную непрерывную область памяти, в которой хранятся некоторые данные. А раз это так, то изменение значений в этих ячейках автоматически приведет к изменению значения самой переменной. Но как мы можем изменить эти ячейки, не используя саму переменную? Вот как раз это способны делать указатели. Именно для этого они существуют – для записи и считывания данных из произвольных ячеек памяти. Это еще называется **прямым доступом к памяти**.

Так что же из себя представляют указатели? Возможно, вас это удивит, но это, по сути, те же самые переменные, причем, целочисленные. А хранят они (в виде целого числа) адрес той ячейки памяти, в которую можно что-то записать или прочитать, то, что в ней содержится. Поэтому размер данных указателя в памяти определяется разрядностью системы. Если система 32-х разрядная, то указатель занимает 4 байта; если 64-х разрядная, то 8 байт:



Этого как раз достаточно, чтобы охватить все ячейки памяти целым беззнаковым значением. Соответственно, для систем с другими разрядностями, размер указателей также будет другим. Но, чаще всего, мы имеем дело с 32-х и 64-х разрядами.

Давайте теперь посмотрим, как в языке Си можно объявить указатель. Общий синтаксис здесь такой:

<тип данных> *<имя указателя>;

Вначале указываем тип данных, с которым будет работать указатель. Затем ставится звездочка и следом прописывается имя указателя. Обратите внимание, сам по себе указатель всегда хранит лишь адрес переменной и имеет фиксированный размер (4 байта для 32-х разрядных систем; 8 байт для 64-х разрядных систем). Тип, который мы указываем вначале, относится не к указателю, а к типу данных, с которыми его предполагается использовать. Сейчас на конкретных примерах вы все это увидите и поймете.

Давайте предположим, что мы хотим через указатель взаимодействовать с байтовой переменной d. Так как переменная d имеет тип char, то и при объявлении указателя вначале указать этот же тип char:

```
char d = 10;
char *gpt;
```

После этого указателю нужно присвоить адрес переменной (ячейки памяти), где располагается переменная d. В нашем примере – это ячейка с номером 34024. То есть, нужно записать следующую команду:

```
gpt = 34024;
```

В результате указатель gpt будет хранить это целое число, следующим образом:

34024 = 232 + 256 · 132

В первом байте будет записано 232, во втором – 132, а все остальные равны нулю. Как только мы присвоили указателю адрес переменной, то говорят, что указатель **указывает (ссылается)** на эту переменную. После этого мы можем совершенно спокойно посредством указателя gpt записывать значения в ячейку с номером 34024 и считывать оттуда данные. Делается это следующим образом:

```
char x = *gpt;  // считывание значения из ячейки памяти
*gpt = 100;     // запись значения 100 в ячейку памяти
```

То есть, когда перед именем указателя стоит символ *, его следует воспринимать как переменную типа char, расположенной по адресу 34024. Соответственно, команда x = *gpt; будет выполнять чтение данных из этой переменной, а команда *gpt = 100;- запись значения 100 в эту переменную. И, так как переменная *gpt располагается в данной и той же ячейке, то сначала будет прочитано значение 10 и присвоено переменной x, а затем, переменная d будет изменена на значение 100. Сама операция *, записанная перед именем указателя (после его объявления), называется **операцией разыменования**.

Вот принцип объявления и использования указателей в языке Си. Конечно, сейчас у вас может возникать вопрос зачем все это надо? Есть же переменные и с ними можно работать напрямую? Какой смысл в этих указателях? Но, все по порядку. И первый правильный вопрос такой: как нам в реальной программе узнать адрес расположения переменной? Число 34024, что мы использовали – это всего лишь иллюстрация. В реальности, та же переменная d может находиться в любой доступной ячейке памяти. Как же узнать номер этой ячейки? И здесь тоже все очень просто. В языке Си есть специальный оператор, который возвращает адрес переменных. Он определяется символом амперсанд и записывается перед именем переменной. Например, так:

```
gpt = &d;
```

В результате указатель gpt будет содержать адрес переменной d, где бы она ни располагалась в памяти. Это универсальная конструкция. И, обратите внимание, здесь мы прописываем указатель без символа *, так как присваиваем ему адрес. Давайте еще раз. Смотрите, если записать указатель в двух формах:

```
gpt = 0;
*gpt = 0;
```

то в первой строчке присваивается адрес, с которым, затем, этот указатель может работать, а во второй заносим значение 0 в ячейку памяти с адресом 0. Вот эти две формы использования указателей нужно очень хорошо запомнить и понимать, как они работают. Если вам кажется, что вы сейчас что-то не улавливаете, то изучите этот вопрос подробнее, прежде чем идти дальше.

Итак, мы теперь готовы написать свою первую программу с использованием указателей. Пусть она делает ровно то, о чем я только что рассказывал: меняет значение переменной d через указатель gpt:

```
#include <stdio.h>

int main(void)
{
        char d = 10;
        char *gpt;
        gpt = &d;

        printf("gpt = %p, *gpt = %d, d = %d\n", gpt, *gpt, d);

        *gpt = 100;
        printf("gpt = %p, *gpt = %d, d = %d\n", gpt, *gpt, d);

        return 0;
}
```

После выполнения программы увидим строчки:

gpt = 0062ff1b, *gpt = 10, d = 10
gpt = 0062ff1b, *gpt = 100, d = 100

Смотрите, указатель gpt хранит адрес 0062ff1b, где расположена байтовая переменная d со значением 10. Операция *gpt позволяет прочитать значение этой переменной и оно, естественно, совпадает с самой переменной d. После этого с использованием той же операция *gpt мы заносим в ячейку памяти с номером 0062ff1b значение 100. И видим, что переменная d также изменила свое значение.

А теперь, смотрите, мы с вами в программе можем объединить две строчки:

```
char *gpt;
gpt = &d;
```

в одну следующим образом:

```
char *gpt = &d;
```

На первый взгляд, кажется, что мы используем форму записи указателя со звездочкой, а значит, присваиваем ему не адрес, а заносим некоторое значение по какому-то адресу. Но это не так. Здесь происходит операция инициализации указателя, а не присваивания. Когда мы говорили о переменных, то отмечал этот факт. Если в момент объявления переменных мы им сразу что-либо присваиваем, то отрабатывает операция инициализации, которая, в общем случае, отличается от операции простого присваивания. Здесь, как раз тот самый случай. При инициализации указателя ему присваивается адрес, а звездочка записана для объявления переменной gpt как указателя, а не просто как обычной переменной. Инициализатор это просто «понимает» и заносит в указатель адрес переменной d. Этот момент также следует понимать. Он часто используется на практике.

Итак, мы с вами научились объявлять указатель типа char и с его помощью читать и менять значение байтовой переменной. Давайте теперь посмотрим, что будет происходить, если вместо типа char прописать тип int и у переменной и у указателя:

```
#include <stdio.h>

int main(void)
{
        int d = 10;
        int *gpt = &d;

        printf("gpt = %p, *gpt = %d, d = %d\n", gpt, *gpt, d);

        *gpt = 75432;
        printf("gpt = %p, *gpt = %d, d = %d\n", gpt, *gpt, d);

        return 0;
}
```

После запуска увидим строчки:

gpt = 0062ff18, *gpt = 10, d = 10
gpt = 0062ff18, *gpt = 75432, d = 75432

На первый взгляд, работа программы никак не изменилась. Однако тип int обрабатывается несколько иначе. Когда выполняется команда:

```
*gpt = 75432;
```

то на уровне машинных кодов оно автоматически раскладывается по четырем байтам в числа 168, 38, 1, 0:

75432 = 168 + 256 · 38 + 256^2 · 1 + 256^3 · 0

И эти числа, начиная с адреса переменной d, последовательно заносятся в четыре байта, которые определяют значение этой переменной. То есть, здесь тип int, который прописан при объявлении указателя gpt, как раз и указывает компилятору, как следует интерпретировать и в какое количество ячеек заность присваивание данных. Напримерс, если следующей строчкой присвоить число 1:

```
*gpt = 1;
```

то все равно будет сформировано четыре числа:

1, 0, 0, 0

которые будут записаны в соответствующие ячейки переменной типа int.

А вот если бы у нас указатель по-прежнему имел бы тип char, то было бы сформировано только одно число 1 и занесено только в одну ячейку, так как char – это один байт. Вот в чем смысл типа, который прописывается при объявлении указателя. Мы указываем компилятору, в каком формате следует представлять данные и как их заносить или считывать по указанному адресу. Поэтому, при работе с теми или иными переменными через указатели, типы должны совпадать, иначе возможны непредвиденные ошибки или неточности в представлении данных.

На этом мы завершим наше первое занятие по указателям. Мы сделали первый, важный шаг в понимании, что это вообще такое. Увидели, как объявляются указатели и как с их помощью можно менять значения в заданных ячейках памяти, обращаясь к ним напрямую, а не через переменные. На следующем занятии мы продолжим эту тему и окончательно развеем мифы о сложности этой одной из самых простых концепций языка программирования Си.

Практический курс по C/C++: https://stepik.org/course/193691

Видео по теме


Язык Си. Рождение легенды


#1. Этапы трансляции программы в машинный код. Стандарты


#2. Установка компилятора gcc и Visual Studio Code на ОС Windows


#3. Структура и понимание работы программы "Hello, World!"


#4. Двоичная, шестнадцатеричная и восьмеричная системы


#5. Перемен базовые т Модификаторы т