Функции и рекурсия

Сегодня мы изучим функции — части программы, которые можно повторно вызывать с разными параметрами, чтобы не писать много раз одно и то же. Функции в программировании немного отличаются от математических функций. В математике функции могут только получить параметры и дать ответ, в программировании же функции умеют делать что-нибудь полезное.

**Наибольший общий делитель**

Рассмотрим такую задачу: нужно найти наибольший общий делитель двух чисел. Наибольшим общим делителем для двух целых чисел A и B будет такое наибольшее целое число C, на которое и A, и B делятся без остатка. Например, есть две пиццы: одна разрезана на 12 кусков, а другая на 8. Нужно раздать максимальному числу людей одинаковое количество кусочков, как первой, так и второй пиццы. В данном случае ответ будет равен 4 — наибольшему общему делителю чисел 8 и 12.

Пользоваться функциями мы уже умеем. Например, мы делали функцию main, а также пользовались готовыми, так что ничего сложного нас не ждет.

Создавать новые функции нужно между using namespace std и функцией main.

Наша функция должна называться gcd (от английского greatest common divisor, наибольший общий делитель). Функция будет принимать два параметра a и b (числа, наибольший общий делитель которых нужно найти) и возвращать в ответ одно число.

int gcd(int a, int b) {

    while (b != 0) {

        int c = a % b;

        a = b;

        b = c;

    }

    return a;

}

Перед именем функции нужно записать тип возвращаемого значения. В нашем случае это int — целое число. Затем пишется название функции, а после него в скобках через запятую перечисляются параметры функции, перед каждым из них указывается тип. Когда мы будем вызывать функцию, то в качестве параметров можем указывать любое арифметическое выражение — оно будет скопировано в локальные переменные функции a и b. Эти переменные видны только внутри нашей функции, и получить к ним доступ из других функций нельзя.

После этого начинается блок инструкций, как и в main. В нашем случае внутри функций реализован алгоритм Евклида, который изучается в младшей школе. Только вычитание заменяем взятием остатка — так ответ будет находиться намного быстрее. Заканчивается функция командой return, после которой через пробел пишется, что нужно вернуть в качестве ответа. Этот ответ будет подставлен в то место, откуда вызвали функцию.

Теперь посмотрим на вызов функции. Он делается точно так же, как и вызовы стандартных функций.

int main() {

    int n, m;

    cin >> n >> m;

    cout << gcd(n, m);

    return 0;

}

Обратите внимание, что переменные в main и внутри функции называются по-разному. Это абсолютно нормально, ведь при каждом вызове функции происходит копирование переданных значений в локальные переменные функции. И когда наша функция gcd изменяет значения своих локальных переменных a и b, то переменные n и m остаются неизменными.

При отладке функций есть два удобных варианта пошагового выполнения программы —– «шаг с заходом» (step into, кнопка F11), который входит внутрь функции, и «шаг с обходом» (step over, кнопка F10), который не входит внутрь функции, а сразу подставляет на место ее вызова результат.

**Сокращение дроби**

С помощью функции поиска наибольшего общего делителя мы можем решить задачу о сокращении дроби. Например, дробь 12/8 должна превратиться в результате работы нашей программы в несократимую дробь 3/2. Для сокращения дроби нужно найти наибольший общий делитель числителя и знаменателя, а затем разделить их на него.

Напишем функцию, которая будет сокращать дробь. Она принимает на вход два числа и должна возвращать также два числа — новые числитель и знаменатель. Мы пока не умеем возвращать пару чисел, поэтому нам придется научиться изменять значение переданных переменных. Код будет выглядеть так:

void reduce(int &a, int &b) {

    int c = gcd(a, b);

    a /= c;

    b /= c;

}

Здесь void обозначает, что функция не будет возвращать ничего, только выполнять какие-то действия. Перед именами переменных мы поставили значок & (он называется амперсанд). Это означает, что у нас не будет создаваться копия переменных внутри функций. Функция получит возможность изменять переданные ей параметры. Таким образом, в качестве параметра такой функции при вызове могут выступать только переменные, ведь изменить значение арифметического выражения нельзя. Вызов функции должен происходить отдельно, не внутри арифметического выражения. Например, вызов может происходить так:

 int main() {

    int n, m;

    cin >> n >> m;

    reduce(n, m);

    cout << n << " " << m;

    return 0;

}

В этой программе есть уже 3 функции: gcd, reduce и main. Причем функции вызывают друг друга и ничего страшного не происходит: при вызове функции сначала выполняется она, а когда результат подсчитан, выполнение продолжается с того места, где функция была вызвана. Точно также функция может вызывать другие экземпляры себя самой. Это называется рекурсия и тоже совсем не страшно.

**Рекурсия**

Работая с функциями, нужно различать две сущности: последовательность команд, которые выполняются в функции, и локальные переменные конкретного экземпляра функции. При рекурсивном запуске последовательность команд у функций одинаковая, но локальные переменные у каждого экземпляра свои. Кроме этого каждый экземпляр помнит, куда он должен вернуться после завершения работы. Например, при рекурсивном запуске нужно возвращаться в вызвавший эту функцию другой экземпляр той же функции.

Представим себе, что у нас есть миллиард человек (это будущие экземпляры функции), сидящих в ряд, и у каждого из них есть листочек для записи (это его локальная память). Нам нужно произносить числа и написать инструкцию для людей так, чтобы они в итоге сказали все числа из последовательности в обратном порядке. Пусть каждый из них будет записывать на своем листочке только одно число. Тогда инструкция для человека будет выглядеть так:

1)     Запиши названное число

2)     Если число не последнее – потереби следующего за тобой человека, пришла его очередь работать

3)     Когда следующий за тобой человек сказал, что он закончил – назови записанное число

4)     Скажи тому, кто тебя теребил (предыдущий человек), что ты закончил

Формализуем задачу. Пусть задается последовательность натуральных чисел, заканчивающаяся нулем. Необходимо развернуть ее с помощью рекурсии.

void rec() {

    int n;

    cin >> n;

    if (n != 0) {

        rec();

        cout << n;

    }

}

Наша функция не принимает параметров. Чтобы запустить разворот последовательности, нужно вызвать функцию один раз из main.

**Факториал**

Факториалом числа N называется произведение всех чисел от 1 до N. Например, 4! = 1×2×3×4 = 24. Тем, кто знаком с методом математической индукции, будет довольно просто осознать рекурсию. Как и в математической индукции, в рекурсии должна быть база (момент, когда функция не вызывает другую рекурсивную функцию) и переход (правило, по которому считается результат по известному результату для меньшего параметра). Наша функция подсчета факториала делает только свою работу, но пользуется результатами чужого труда. Например, если функция получила на вход параметр 4, то должна вернуть 4 умноженное на 3! (который будет посчитан другими функциями). В случае факториала аналогом «базы индукции» может выступать 0! — по определению он равен единице. Функция будет выглядеть следующим образом:

int fact(int n) {

    if (n == 0) {

        return 1;

    }

    return n \* fact(n – 1);

}

Обратите внимание, что после выполнения команды return функция заканчивает свою работу, поэтому else писать не обязательно. При n равном нулю, выполнится return 1 и работа функции будет завершена — до следующего return выполнение просто не дойдет.

Этот пример иллюстрирует общую схему написания рекурсивных функций: сначала проверяется условие, когда функция должна закончиться, а дальше делается все остальное. При этом параметр должен сходиться к значению базы. Обычно это означает, что при каждом следующем вызове рекурсии параметр должен уменьшаться.

**Число сочетаний**

С помощью функции подсчета факториала можно решить еще одну полезную задачу — посчитать число сочетаний. Пусть у нас есть N человек и из них нужно выбрать K добровольцев. Нужно посчитать, сколькими различными способами можно это сделать. Называется это «C из N по K». Из математики известен замечательный факт, что C из N по K равно N! / (K! × (N - K)!).

Это можно записать в виде функции:

int cnk(int n, int k) {

    return fact(n) / (fact(k) \* fact(n – k));

}

**Функции, возвращающие логическое значение**

Иногда удобно оформлять даже простые вещи в виде функций, чтобы повысить читаемость программы. Например, если нужно проверить число на четность, то гораздо понятнее будет каждый раз вызывать функцию is\_even(n), а не писать каждый раз n % 2 == 0.

Такая функция может выглядеть так:

bool is\_even(int n) {

    return n % 2 == 0;

}

Результатом работы этой функции будет истина или ложь. Теперь функцию очень удобно применять в if’ах:

if (is\_even(n)) {

    cout << "EVEN";

} else {

    cout << "ODD";

}

 Если есть сложное логическое выражение, то лучше оформить его в виде функции с говорящим названием — так программу будет легче читать, а вероятность ошибок в ней резко снизится.

**Философские вопросы применения функций**

Функции чрезвычайно полезны, если одни и те же действия нужно выполнять несколько раз. Но некоторые логические блоки работы с программой иногда тоже удобно оформлять в виде функции. Это связано с тем, что человек может одновременно держать в голове ограниченное количество вещей. Когда программа разрастается, отследить все уже очень сложно. В пределах одной небольшой функции запутаться гораздо сложнее — известно, что она получает на вход, что должна выдать, а об остальной программе в это время можно не думать.

В программировании также считается хорошим стилем писать функции, умещающиеся на один экран. Тогда можно одновременно окинуть взглядом всю функцию и не нужно крутить текст туда-сюда. Поэтому, если получилось что-то очень длинное, то нужно нарезать его на кусочки так, чтобы каждый из них был логичным (делал какое-то определенное действие, которое можно назвать) и не превышал при этом 10-15 строк.