1 Занятие №2.

1.1 Первая программа

Рассмотрим такую задачу: *написать программу, которая вводит пары чисел и вычисляет их НОД (наибольший общий делитель)*. Для вычисления будем использовать соотношение

$$\gcd(a,b) = \gcd(b, a \bmod b).$$

1.1.1 Текст программы

Полный текст программы приведён ниже.

```
#include <stdio.h>

int main(void)
{
        int a, b;
        int c;

        while (scanf("%d%d", &b, &c) == 2) {

                /* invariant: GCD(a,b)==GCD(b,a%b) */

                do {
                        a = b;
                        b = c;
                        c = a % b;
                } while (c != 0);
                printf("%d\n", b);
        }

        return 0;
}
```

Теперь мы разберём его по шагам, демонстрируя базовые возможности языка Си.

1.1.2 Директива препроцессора

```
#include <stdio.h>
```

Это — так называемая директива препроцессора. Перед основной фазой трансляции в текст программы будет добавлено содержимое файла `stdio.h`, которые находится в одном из стандартных каталогов операционной системы. Собственно язык Си не определяет никаких функций, которые поддерживают операции ввода/вывода, работу со строками и другие необходимые возможности. Все они вынесены в стандартную библиотеку Си. В UNIX-системах стандартную библиотеку языка Си часто называют **libc**. Файл `stdio.h` — это один

из файлов стандартной библиотеки, в котором определяются типы данных, константы, переменные и функции, необходимые для операций ввода/вывода. Наличие такой директивы препроцессора указывает, что в программе будут использоваться функции ввода/вывода. Поскольку почти всякая программа что-либо вводит или выводит, включение файла `stdio.h` будет присутствовать почти в каждой программе.

1.1.3 Функция `main`

```
int main(void)
{

}
```

Это — определение функции с именем `main`. В языке Си отсутствует деление на процедуры и функции, все выполняемые объекты называются функциями, даже если они ничего не возвращают.

Программа на языке Си является совокупностью функций и глобальных определений, то есть понятие о каком-то главном блоке, с которого начинается работа, отсутствует. В программе должна быть определена функция с именем `main`, именно эта функция будет вызвана первой, после того, как завершится системная инициализация.

В данном определении функция `main` не принимает никаких параметров и возвращает целое значение. Поскольку функция `main` вызывается операционной средой, тип параметров и тип возвращаемого значения достаточно жёстко фиксированы. Никогда не определяйте функцию `main` как не возвращающую значения!

Фигурные скобки после заголовка функции — это составной оператор, аналог **`begin`** и **`end`** языка Паскаль. Тело функции в языке Си заключается в составной оператор, даже если оно состоит из одного оператора. Иногда составной оператор мы будем просто называть *блоком*.

В начале любого составного оператора, а не только того, которые образует тело функции, могут стоять определения переменных. Эти переменные, как правило, существуют ограниченное время, пока составной оператор не завершил работу.

В C90[1] все определения локальных переменных должны размещаться до операторов, а в стандарте C99[2], как и в языке C++, определения локальных переменных и операторы могут быть перемешаны.

1.1.4 Определения переменных

```
int a, b;
int c;
```

Это — определение локальных переменных `a`, `b`, `c`, которые имеют целый тип.

Простейшее определение переменных имеет такой вид.

```
определение = тип список_переменных ";"
список_переменных = переменная { "," переменная }
переменная = имя [ "=" значение ]
```

При определении переменная может быть сразу проинициализирована некоторым значением, например.

[1] ISO стандарт языка Си, принятый в 1990 году. Этот стандарт (с поправками) — именно то, что обычно называется ANSI C.

[2] Новый стандарт языка Си, принятый, очевидно, в 1999 году.

char	целый тип, достаточный для хранения кода символа.
signed char	знаковый целый тип, достаточный для хранения символа.
unsigned char	беззнаковый целый тип, достаточный для хранения символа.
[signed] short [int]	короткое целое со знаком.
unsigned short [int]	короткое целое без знака.
[signed] int	целое.
unsigned [int]	целое без знака.
[signed] long [int]	длинное целое.
unsigned long [int]	длинное целое без знака.
[signed] long long [int]	более длинное целое (введён стандартом C99).
unsigned long long [int]	более длинное целое без знака (введён стандартом C99).
float	число с плавающей точкой одинарной точности.
double	число с плавающей точкой двойной точности.
long double	число с плавающей точкой расширенной точности.

Таблица 1: Простые типы языка Си

```
int a = 4;
```

Регистр букв (заглавные—строчные) в идентификаторах и ключевых словах значим, то есть `Main` и `main` — это два различных идентификатора. Тем не менее, в программе не рекомендуется использовать имена, различающиеся только регистром букв.

1.1.5 Простые типы данных

В языке Си определены 4 основных простых типа.

char целый тип, достаточный для хранения символов;
int целый тип;
float вещественное число с одинарной точностью;
double вещественное число с двойной точностью.

Чтобы определить тип более точно используются квалификаторы **short**, **long**, **signed**, **unsigned**, в результате можно использовать следующие простые типы данных, которые перечислены в таблице 1.

Ключевое слово, записанное в квадратных скобках, означает, что оно может быть опущено. Так, тип **[signed] long [int]** может записываться и как **signed long**, и как **long int**, и как **signed long int**, и просто как **long**. Кроме того, ключевые слова могут свободно переставляться друг относительно друга, то есть тот же самый тип может задаваться и как **long int signed**.

Является ли тип **char** знаковым или беззнаковым, зависит от конкретной реализации компилятора языка Си. Поэтому везде, где требуется выполнять арифметические операции, зависящие от знаковости, необходимо явное указание знаковости типа **char**.

Размер типа **char** является единицей измерения размера всех других типов. Постулируется, что переменная типа **char** занимает один байт. Таким образом может получиться, что байт будет содержать, например, 9 битов. Все прочие типы имеют размер, кратный целому числу байтов.

Тип **int** имеет размер машинного слова на данной архитектуре. Если машинное слово — 16 бит, то и тип **int** будет иметь такой же размер.

Все знаковые типы имеют точно такой же размер, как и беззнаковые типы. Способ хранения знаковых чисел стандартом неопределён, но все существующие архитектуры хранят знаковые числа в формате дополнения до 2.

Стандарт определяет следующие неравенства для размеров целых типов.

$$sizeof(char) < sizeof(short) \leqslant sizeof(int) \leqslant sizeof(long) \leqslant sizeof(long\ long)$$

$$sizeof(short) < sizeof(long)$$

$$sizeof(int) < sizeof(long\ long)$$

Все современные архитектуры имеют байт, состоящий из 8 бит. На многих современных 32-битных архитектурах целые типы имеют следующие размеры:

short 2 байта (16 бит).
int 4 байта (32 бита).
long 4 байта (32 бита).
long long 8 байтов (64 бита).

Для типов с плавающей точкой точность (количество битов мантиссы) и диапазон представления (количество битов порядка) растёт от **float** до **long double**.

1.1.6 Функция ввода данных

```
scanf("%d%d", &b, &c)
```

Это — вызов функции `scanf` для считывания данных со стандартного потока ввода (обычно стандартный поток ввода — это клавиатура). Функция `scanf` первая функция стандартной библиотеки языка Си, которую мы рассмотрим.

Первый аргумент функции это строка формата (строки мы будем рассматривать позже, а пока заметьте, что они записываются в кавычках), которая определяет, значения каких типов должны быть считаны. Остальные аргументы задают переменные, в которые должны быть считаны значения.

Строка формата состоит из спецификаторов ввода полей. Каждый спецификатор ввода начинается со знака `%` («процент»). Простейшие спецификаторы ввода перечислены ниже.

`%d`	Считать целое число (**`int`**). Перед чтением числа пропускаются все пробельные символы (пробелы, табуляции, переводы строк). Если первый непробельный символ не может начинать число, функция `scanf` завершается. Иначе число считывается либо пока не возникнет переполнения, либо пока не встретится символ, который не может быть частью числа. Если возникло переполнение, функция `scanf` завершается с неудачей. Если встретился нецифровой символ, чтение числа считается успешным, а этот символ не будет считан из потока.
`%ld`	Считать длинное целое число (**`long int`**). Используются те же самые правила, что и при чтении обычного целого числа.
`%Ld`	Считать длинное целое число (**`long long int`**). Используются те же самые правила, что и при чтении обычного целого числа.
`%f`	Считать вещественное число типа **`float`**. Применяются те же правила, что и при чтении целых чисел.
`%lf`	Считать вещественное число типа **`double`**. Применяются те же правила, что и при чтении целых чисел.
`%Lf`	Считать вещественное число типа **`long double`**. Применяются те же правила, что и при чтении целых чисел.
`%c`	Считать очередной символ из входного потока в переменную типа **`char`**.

Несколько спецификаций формата могут быть записаны в одной форматной строке. Например, `"%lf%d%f"` — считать вещественное значение в переменную типа `double`, затем целое значение в переменную типа `int`, затем вещественное значение в переменную типа `float`.

После спецификации формата перечисляются переменные, в которые будет записано значение. Для всех простых типов, которые были упомянуты выше, перед именем переменной *обязательно* должен стоять знак `&`. Это — унарная операция взятия адреса переменной, которую мы ещё рассмотрим в дальнейшем.

Количество спецификаций формата в форматной строке обязательно должно совпадать с количеством переменных, указанных после неё. Кроме того, тип, указанный в спецификации формата, обязательно должен совпадать с типом соответствующей переменной. Если вы где-то ошиблись, то многие компиляторы ничего не заметят и скомпилируют программу. При выполнении такая программа будет либо давать неправильный результат, либо вообще завершаться аварийно.

Функция `scanf` возвращает количество успешно считанных полей, заданных в спецификации. В примере из программы, которую мы пишем, значение 2 означает, что успешно считаны два целых числа, 1 означает, что было считано только первое число, а второе — нет (закончился файл или до начала чтения числа встретился символ, которые не может быть частью целого числа). Если бы `scanf` вернул 0, это значит, что ни одна переменная не была считана из-за неверного задания входных данных в файле. Если данные закончились (то есть чтение дошло до конца файла) до того, как была считана хотя бы одна переменная, функция `scanf` вернёт значение, задаваемое константой `EOF`.

Игнорировать значение, возвращаемое функций `scanf` нельзя! То есть, язык, конечно, позволяет это делать, но программа, написанная таким образом будет работать некорректно, если пользователь ошибся при вводе.

1.1.7 Оператор цикла `while`

```
while (scanf("%d%d", &b, &c) == 2) {

}
```

Это — оператор цикла **while**. Цикл **while** выполняется (как и в Паскале) до тех пор, пока истинно выражение, записанное в скобках. В данном случае цикл будет выполняться, пока функция `scanf` возвращает значение 2, то есть пока считываются оба числа `b` и `c`. Круглые скобки здесь — часть оператора цикла **while**, а не выражения условия цикла. Круглые скобки не могут быть пропущены.

Телом цикла **while** может быть любой оператор и, в частности, составной оператор, как в разбираемой программе.

1.1.8 Операции сравнения и логические операции

В языке Си определены обычные операции сравнения чисел, которые записываются следующим образом:

`==` сравнение двух чисел на равенство.
`!=` сравнение на неравенство.
`>=` «больше или равно».
`>` «больше».
`<=` «меньше или равно».
`<` «меньше».

Обратите внимание, что сравнение на равенство записывается как два знака равенства ==, а один знак равенства = — это операция присваивания!

Подробнее то, как вычисляются выражения, мы рассмотрим подробнее на следующих занятиях.

Для проверки нескольких условий используются логические операции-связки `||` и `&&`. Обратите внимание, что оба знака операции состоят из двух символов. Есть и операции `|` и `&`, это совсем другие операции, мы их рассмотрим позднее.

Операция `||` — логическое «или». Она даёт истинное значение, если хотя бы один из аргументов даёт истинное значение. При этом операция вычисляется по «короткой» схеме, то есть если первый аргумент операции дал «истину», второй даже не вычисляется.

Операция `&&` — логическое «и». Она даёт значение «ложь», если хотя бы один из аргументов даёт значение «ложь». Как и предыдущая, эта операция вычисляется по «короткой» схеме. Если первый аргумент дал значение «ложь», второй аргумент даже не вычисляется.

Обратите внимание, что в языке Си отсутствует логический тип как таковой. Везде, где требуется логическое значение «ложь» или «истина», может использоваться любое целое выражение (и вообще, любое скалярное выражение). Значение 0 понимается, как «ложь», а любое значение, не равное 0, как «истина». Тем не менее, операции отношения и логические операции вырабатывают в качестве значения «истины» вполне определённое значение — 1.

В новом стандарте **C99** тип **bool** всё-же введёт. Но все правила, описанные выше, продолжают работать.

1.1.9 Оператор do while

```
do {

} while (c != 0);
```

Это — оператор цикла с постусловием. Отличия от цикла **repeat until** языка Паскаль перечислены ниже:

1. Ключевые слова **do** и **while** не образуют блока. Поэтому, если в цикле необходимо записать несколько операторов, нужно использовать составной оператор.

2. Круглые скобки после ключевого слова **while** являются частью оператора цикла, а не выражения, и поэтому обязательны.

3. Цикл выполняется, пока условие, записанное после **while** истинно.

1.1.10 О расстановке «;»

В языке Си «точки с запятой» являются составной частью каждого оператора, кроме составного, поэтому должны обязательно ставится после них. После составного оператора точка с запятой никогда не должна ставиться!

1.1.11 Арифметические выражения и присваивания

```
a = b;
b = c;
c = a % b;
```

В языке определены обычные арифметические операции:

`+` сложение.

`-` вычитание.

`*` умножение.

`/` деление. Если оба операнда операции — целые выражения, деление будет выполняться как деление нацело, результатом будет тоже целое значение. Если хотя бы один операнд — вещественное число, деление будет выполняться над вещественными числами и даст вещественный результат.

`%` взятие остатка от деления. Применимо только к целым выражениям.

При выполнении операций сложения и вычитания с целыми числами арифметическое переполнение или переносы не диагностируются. Программа продолжит работать как ни бывало. Но при выполнении операций умножения и деления, а также сложения и вычитания с вещественными числами, ошибки диагностируются и вызывают аварийное завершение программы.

Операция присваивания = даёт результат, равный присвоенному значению, поэтому допустимо использование нескольких присваиваний подряд, например:

```
a = b = c = 0;
```

1.1.12 Функция вывода данных

```
printf("%d\n", b);
```

Функция `printf` печатает значения на стандартный поток вывода (обычно это экран). Первый параметр функции — это строка формата печати. Строка формата может содержать обычные литеры, которые будут просто печататься, а может содержать спецификации формата печати, похожие на рассмотренные нами при разборе функции `scanf`. Простейшие из них перечислены ниже.

`%d` печать целого (**`int`**) значения со знаком.
`%u` печать беззнакового целого (**`unsigned int`**).
`%ld` печать длинного целого (**`long`**) со знаком.
`%lu` печать беззнакового длинного целого (**`unsigned long`**).
`%Ld` печать длинного целого (**`long long`**) со знаком.
`%Lu` печать беззнакового длинного целого (**`unsigned long long`**).
`%c` печать литеры.
`%f` печать вещественного числа (**`double`**).
`%Lf` печать вещественного числа (**`long double`**).

В форматной строке могут использоваться специальные последовательности символов: `\n` — переход на следующую строку, `\t` — символ табуляции (для равного отступа столбцов). Чтобы напечатать знак процента, он повторяется дважды.

Знак взятия адреса `&` для всех пока рассмотренных нами типов не ставится! Аргументы печати должны быть точно того типа, который указан в спецификации формата (исключения — см. ниже). Они должны идти в том же порядке, в котором перечислены в строке формата. В противном случае ваша программа будет в лучшем случае печатать что-то странное, а в худшем — аварийно завершаться.

Для печати значений типа **`short`** или **`signed char`** нужно использовать спецификатор печати чисел типа **`int`**. Для печати значений типа **`unsigned short`** или **`unsigned char`** нужно использовать спецификатор печати чисел типа **`unsigned int`**. Для печати значений типа **`float`** используется спецификатор печати значений типа **`double`**.

1.1.13 Возврат значения из функции

```
        return 0;
```

Поскольку функция `main` объявлена как возвращающая целое значение, необходим оператор, который определит возвращаемое значение. Оператор **`return`** вызывает завершение работы функции и возврат значения, указанного в операторе.

Относительно самого возвращаемого значения пока заметим, что функция `main` в обычных случаях должна возвращать значение 0.

1.2 Вторая программа

Рассмотрим следующую задачу: *удалить из входного потока все пробельные литеры*.

```
#include <stdio.h>

int main()
{
        char c;

        while (scanf("%c", &c) == 1) {
                if (c != ' ') printf("%c", c);
        }
        return 0;
}
```

1.2.1 Условный оператор

Условный оператор имеет следующий синтаксис:

```
if ( выражение ) оператор [ else оператор ]
```

Отличия от соответствующего оператора языка Паскаль следующие:

1. Круглые скобки являются частью оператора **`if`**, и не могут быть пропущены.
2. Ключевое слово **`then`** отсутствует.

1.2.2 Символьные константы

Замечание: слово «символ» — очень сильно перегружено разными смыслами, поэтому иногда употребляют слово «литера».

Литерные константы записываются в апострофах (одинарных кавычках). Например, `'a'` — литера `a`. Кроме того, для специальных символов применяется следующая запись: `'\''` — литера «апостроф», `'\\'` — литера «обратная косая черта» (backslash), `'\n'` — литера перехода на новую строку, `'\t'` — литера табуляции.

Обратите внимание, что литерные константы имеют тип **`int`**, а не **`char`**.