

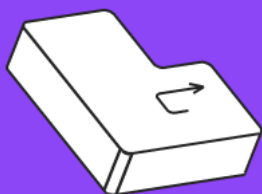


Урок 5

Функции

Буферный ввод

Программирование на языке Си
(базовый уровень)



Оглавление

Введение	2
Термины, используемые в лекции	3
Функции	4
Определение и вызов функции	4
Прототип или описание функции	6
Параметры и результат функции	7
Задачи	9
Функции и область видимости	9
Буферный ввод	12
Потоки ввода и вывода	12
ASCII	13
Функции ввода-вывода <code>getchar()</code> и <code>putchar()</code>	15
Функции не буферизированного ввода <code>getch()</code> и <code>getche()</code>	18
Оформление функций	18
Вычисление корней квадратного уравнения	19
Дорабатываем систему меню	19
Добавляем функции	20
Рефакторинг	21
Пример применения <code>static</code> -переменных для диагностики	21
Машины состояний и конечные автоматы	22
Конечный автомат для варки кофе	22
Рефакторинг программы конечного автомата для варки кофе	25
Подведение итогов	26
Дополнительные материалы	26
Используемые источники	27

Введение

На предыдущей лекции вы:

- Рассмотрели оператор выбора `switch`
- Узнали, что такое циклы и какие типы циклов бывают
- Изучили оператор перехода `goto`

- Поняли, что такое область видимости переменных
- Разобрали, как правильно именовать переменные

На этой лекции вы узнаете:

- Что такое функции
- Как создать свою функцию и передавать аргументы в нее
- Что такое ASCII - таблица символов
- Что такое буферизированный и не буферизированный ввод
- Про функции ввода-вывода `getchar()`, `putchar()` и `getch()`
- Модификатор `static` переменных
- Вычисление корней квадратного уравнения: добавляем функции и рефакторим код
- Пример машины состояний: “Конечный автомат для варки кофе”

Термины, используемые в лекции

Функция — фрагмент программы, описывающий решение некоторой подзадачи.

Заголовок функции — определяет имя функции, ее тип и аргументы поступающее ей на вход.

Тело функции — определяет алгоритм, который выполняет данная функция над входными аргументами.

Прототип — это заголовок функции, не содержащий тела функции.

Результат функции — некоторое значение, которое оно возвращает во внешний код.

Буфер — это область памяти, используемая для временного хранения данных при вводе или выводе.

`stdin` — стандартный поток ввода.

`stdout` — стандартный поток вывода.

`stderr` — стандартный поток ошибок.

Конечный автомат — математическая абстракция, модель дискретного устройства, имеющего один вход, один выход и в каждый момент времени находящегося в одном состоянии из множества возможных.

Рефакторинг — это процесс изменения кода, призванный упростить его обслуживание, понимание и расширение, при этом не изменяя его поведение.

Функции

Под **функцией** подразумевается специальным образом оформленный фрагмент программы, описывающий решение некоторой подзадачи, обычно, неоднократно выполняемой при работе программы. По сути функция представляет именованный кусок кода, который можно многократно использовать в различных частях программы.

Использование функций при разработке программ позволяет получить более наглядный, компактный и легче понимаемый код программы, а также облегчает повторное использование кода за счет использования в программах уже ранее описанных функций. Тем самым устраняется дублирование работы и ускоряется процесс разработки программы.

Определение и вызов функции

Формальное определение функции выглядит следующим образом:

```
тип имя_функции(параметры) // Заголовок функции
{
    выполняемые_инструкции // Тело функции
}
```

Функция состоит из двух частей - **заголовок** и **тело**.

Заголовок - определяет имя функции, ее тип и параметры (аргументы) поступающие ей на вход.

Вначале указывается возвращаемый тип функции. Если функция не возвращает никакого значения, то используется тип *void*.

Затем идет имя функции, которое представляет произвольный идентификатор, к которому применяются те же правила, что и к именованию переменных.

После имени функции в круглых скобках идет перечисление параметров.

Тело - определяет алгоритм, который выполняет данная функция над входными аргументами.

Результат работы функции передается в вызываемую программу оператором **return**.

Каждая программа на языке Си должна содержать как минимум одну функцию *main*. Это точка входа, с которой начнется выполнение кода.

Пример функции вычисления максимума двух целых чисел:

<pre>int max (int a, int b) { if (a > b) return a; else return b; }</pre>	<pre>int max (int a, int b) { if (a > b) return a; return b; }</pre>
--	---

Для выполнения функции ее необходимо вызвать. Вызов функции осуществляется по форме:

имя_функции (аргументы) ;

После имени функции указываются скобки, в которых перечисляются аргументы - значения для параметров функции.

Пример вызова функции из программы:

<pre>#include <stdio.h> int max (int a, int b) { if (a>b) return a; return b; } int main() { int a,b; scanf("%d%d",&a,&b); printf("max = %d\n", max(a,b)); return 0; }</pre>



Наличие скобок обязательно даже в случае, когда функция не имеет параметров.

Если функция возвращает значение, то его можно присвоить переменной.

<pre>#include <stdio.h> int max (int a, int b)</pre>
--

```

{
    if (a>b)
        return a;
    return b;
}
int main()
{
    int a,b;
    scanf("%d%d",&a,&b);
    int maximum = max(a,b);
    printf("max = %d\n", maximum);
    return 0;
}

```

Прототип или описание функции

Вызов функции должен происходить после ее определения, как в случае выше. Если мы сначала вызовем, а потом определим функцию, то мы получим ошибку или предупреждение на этапе компиляции.



Компилятор должен знать о функции до ее вызова.

Очень часто бывает необходимо вызывать не реализованные ранее функции, например, из других модулей (файлов). В этом случае по определению функции в программе должно предшествовать ее объявление в виде **прототипа** данной функции. **Прототип** — это заголовок функции, не содержащий тела функции.

Указывается: имя функции, тип возвращаемого ею значения и формальные параметры функции. В прототипе функции можно не указывать имена переменных, а только типы, как показано в коде слева — это сокращенная, удобна для простых функций. Справа же указаны и имена, и типы параметров, такая запись более понятна.

```

#include <stdio.h>
int max (int, int);
int main()
{
    int a,b;
    scanf("%d%d",&a,&b);
    printf("max = %d\n",
max(a,b));
    return 0;
}
int max (int a, int b)
{
    if (a>b)

```

```

#include <stdio.h>
int max (int a, int b);
int main()
{
    int a,b;
    scanf("%d%d",&a,&b);
    printf("max = %d\n",
max(a,b));
    return 0;
}
int max (int a, int b)
{
    if (a>b)

```

<pre> return a; return b; </pre>	<pre> return a; return b; </pre>
----------------------------------	----------------------------------

Параметры и результат функции

Функция может принимать параметры, передаваемые для обработки. Параметры перечисляются в скобках после имени функции имеют следующее определение:

тип имя_функции (тип параметр1, тип параметр2, ... тип параметрN)



Внимание! Если функция не имеет параметров, т.е. мы в нее ничего не передаем, то в качестве списка ее параметров нужно указывать **void**

Например слева функция не получает параметры и не выдает значения, справа только выдает:

<pre> void max (void) { int a,b; //локальные переменные scanf ("%d%d", &a, &b) ; if (a>b) printf ("%d\n", a) ; else printf ("%d\n", b) ; } </pre>	<pre> int max (void) { int a,b; // ВИДНЫ ТОЛЬКО внутри scanf ("%d%d", &a, &b) ; if (a>b) return a; else return b; } </pre>
--	---

Без использования ключевого слова **void** в объявлении *max()* компилятор не сможет обнаружить ошибку, если вы вызываете свою функцию с параметрами, например *max(42, "Hello world, blin")*; Ошибка будет проходить незаметно, и код все равно будет выполняться.

Мы можем в функции изменить значение параметров, однако подобное поведение не всегда желательно. Если в вызов функции передается переменная, то функция получает копию ее значения. Поэтому все манипуляции со значением параметра внутри функции никак не затронут оригинальную переменную.



Внимание! Параметры в Си передаются ТОЛЬКО по значению.

```
#include <stdio.h>
void mult2(int x)
{
    x = 2 * x;
    printf("mult2 func: x = %d \n", x);
}
int main(void)
{
    int a = 35;
    mult2(a);
    printf("main func: a = %d \n", a);
}
```

Для обеспечения возможности изменять значения передаваемых в качестве параметров объектов необходимо использовать **указатели**. Их мы будем изучать на следующих занятиях.

Нередко бывают ситуации, когда надо гарантировать, что параметр сохранит свое значение на протяжении всей работы функции. В этом случае можно объявить параметр как константный с помощью ключевого слова **const**. Например:

```
void change(const int n)    //параметр n нельзя изменить
```

Функция может иметь **результат** — некоторое значение, которое оно возвращает во внешний код. Для возвращения результата функция применяет оператор *return*, который используется следующим образом:

```
return;
return выражение;
```

Если функция не возвращает никакого значения, то в качестве типа возвращаемого значения указывается *void*.

Оператор *return* в таких функциях может отсутствовать, либо присутствовать, но не содержать никакого выражения. Например, функция слева возвращает значение, а справа нет:

```
int max (int a, int b)
{
    if (a > b)
        return a;
    return b;
}
```

```
void max (int a, int b)
{
    if (a > b)
    {
        printf("%d\n", a);
        return;
    }
    printf("%d\n", b);
}
```


Задачи

1. Составьте функцию, модуль числа и приведите пример её использования

```
#include <stdio.h>
int abs(int num)
{
    return (num<0)?-num:num;
}
int main()
{
    int num;
    scanf("%d",&num);
    printf("%d",abs(num));
    return 0;
}
```

2. Опишите функцию, проверяющую, является ли целое неотрицательное число n простым. В случае положительного ответа функция возвращает значение 1, в противном случае — 0

```
int prime(int n)
{
    int i=2;
    while (i*i<=n){
        if (n%i==0)
            return 0;
        i++;
    }
    return 1;
}
```

- 3.

Функции и область видимости

Рассмотрим пример.

```
#include <stdio.h>
void func(void) {
    int a = 5; //локальная
    переменная
    a++;
    printf("a = %d\n",a);
}
int main(void)
{
    func();
    func();
}
```

```
a = 6
a = 6
a = 6
```

<pre>func(); }</pre>	
----------------------	--

Переменная *a* в функции *func* является локальной, это значит, что она видна только внутри функции. Каждый раз при вызове функции она автоматически создается заново и в нее заносится начальное значение 5. После чего это значение увеличивается на 1 и происходит печать.

Рассмотрим еще один пример.

<pre>#include <stdio.h> void func(void) { static int a = 5; //статическая память a++; printf("a = %d\n", a); } int main(void) { func(); func(); func(); }</pre>	<pre>a = 6 a = 7 a = 8</pre>
---	------------------------------

В данном примере добавлен спецификатор *static* перед объявлением переменной *a*. Это означает, что место под переменную будет выделено один раз в статической памяти и она будет проинициализирована только один раз. Это по сути *глобальная* переменная с *локальной* областью видимости.

Что будет напечатано?

<pre>#include <stdio.h> int y = 1; int f(int a) { static int x = 3; x += y; return x + a + y; } int main(void) { int n = 10; y++; printf("Answer = %d\n", f(n)); printf("Answer = %d\n", f(n)); }</pre>	<pre>Answer = ? Answer = ?</pre>
---	----------------------------------



Ответ:

Answer = 17

Answer = 19

Задачи для самостоятельной работы:

1. Составить функцию, модуль числа и привести пример ее использования.
2. Составить функцию возведения числа n в степень p .
3. Описать функцию, которая проверяет, является ли целое неотрицательное число n простым. В случае положительного ответа функция возвращает значение 1 и 0 в противном случае.
4. Описать функцию, которая проверяет, является ли целое неотрицательное число n степенью 3. В случае положительного ответа функция возвращает значение 1 и 0 в противном случае.
5. Пусть n – целое неотрицательное число. Описать функцию от параметра n , которая находит максимальную цифру в числе.
6. Пусть n – целое неотрицательное число. Описать функцию от параметра n , которая находит количество четных цифр в числе
7. Программа. На стандартном потоке ввода задано число n ($n > 0$) и последовательность из n целых чисел. Описав соответствующую функцию, найти количество элементов последовательности, являющихся палиндромами.

Задачи выше будут разобраны на семинаре.

Буферный ввод

В программах выше при вводе данных с клавиатуры мы можем добавлять и удалять символы, и только после нажатия на клавишу **<Enter>** данный блок становится доступен программе. Это происходит потому, что введенные символы накапливаются и хранятся в определенной области памяти, называемой **буфер**. В данном случае ввод с клавиатуры представляет из себя построчный буферный ввод. Буфер передается в программу когда на вход подается символ новой строки — `'\n'`.

```
#include <stdio.h>
int main()
{
    char s[100];
    scanf("%s", s);
    printf("%s\n", s);
    return 0;
}
```

Если выполнять данную программу, то при наборе в консоле **HH<Backspace>ee<Backspace>lll<Backspace>oo<Backspace><Enter>**, текст будет отображаться на экран только после нажатия клавиши **<Enter>**, мы увидим *Hello*. Это характерный признак буферного ввода.

Если же текст будет отображаться немедленно, то результат был бы: *HHeelloo*. Такое поведение характерно для не буферизированного ввода.

В стандарте ANSI C используется буферизированный ввод.

Для чего нужен буферизированный ввод?

- Пересылка нескольких символов в виде одного блока занимает меньше времени, чем посимвольная пересылка.
- В случае опечатки можно отредактировать введенные данные.

Для чего нужен небуферизированный ввод?

- В различных интерактивных программах или играх обычно используется такой тип ввода.

Потоки ввода и вывода

При вводе и выводе программа на языке Си имеет дело с потоками данных в которых передаются входные и выходные данные. Для унификации это

реализовано через механизм файлов. Процесс открытия файла фактически соответствует какому-то из потоков.

Си рассматривает устройства ввода и вывода как обычные файлы размещенные во внешней памяти. Клавиатура и монитор, в частности, рассматриваются как два различных файла автоматически открывающихся при запуске программы на Си.

Когда программа начинает выполняться, по умолчанию открываются три файла `stdin`, `stdout` и `stderr`

Ввод данных ассоциирован с потоком **`stdin`**, а вывод с потоком **`stdout`**, есть еще поток **`stderr`** для вывода ошибок.

Для управления вводом-выводом с помощью этих потоков используются ряд функций:

- **`getchar()`** — ввод с клавиатуры одного символа
- **`putchar()`** — вывод на консоль одного символа
- **`fgets()`** — ввод одной строки
- **`puts()` / `fputs()`** — вывод одной строки на консоль
- **`scanf()`** — ввод с консоли с форматированием данных
- **`sscanf()`** — ввод с из строки с форматированием данных
- **`printf()`** — вывод с форматированием данных

ASCII

Цифровое устройство по умолчанию не понимает символы — только числа. Поэтому буквы, цифры и знаки приходится кодировать, чтобы задавать компьютеру соответствие между определенным начертанием и числовым значением.

ASCII (American standard code for information interchange) — название таблицы (кодировки или набора), в которой печатным и непечатным (управляющим) символам сопоставлены числовые коды, как упоминалось в первой лекции.

Первые две строчки таблицы — управляющие символы: Backspace, перевод строки, начало и конец абзаца и прочие.

В третьей строке расположены знаки препинания и специальные символы, такие как процент % или астериск*.

Четвертая строка — числа и математические символы, а также двоеточие, точка с запятой и вопросительный знак.

Пятая и шестая строчка — заглавные буквы, а также некоторые другие особые символы.

Седьмая и восьмая строки описывают строчные буквы и еще несколько символов.

ASCII Code Chart																
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Этот формат данных может декодировать любая из существующих компьютерных систем. Таблица ASCII определяет коды для символов:

Тип символа	Интервал в DEC	Интервал в HEX
Управляющие символы	[0,32] и 127	[0,0x1F] и 0x7F
Десятичные цифры	[48,57]	[0x30,0x39]
Буквы латинского алфавита заглавные строчные	[65,90] [97,122]	[0x41,0x5A] [0x61,0x7A]
Буквы национального алфавита	[128,255]	[0x80,0xFF]

Давайте рассмотрим структурные свойства таблицы.

1. Коды символов **цифр** «0»-«9» начинаются с 0x30, а заканчиваются кодом числа
 Например, число 5, 0x35 — символ «5». Зная об этом, можно преобразовать **двоично-десятичные числа** (BCD) в ASCII-строку с помощью простого добавления слева 0x30 к каждому двоично-десятичному полубайту.
2. **Буквы** «A»-«Z» верхнего и нижнего регистров различаются в своем представлении только одним битом, что упрощает преобразование регистра и проверку на принадлежность кода к диапазону значений.
3. Буквы представляются своими порядковыми номерами в **алфавите**, записанными пятью цифрами в **двоичной системе счисления**, перед

которыми стоит 010₂ (для букв верхнего регистра) или 011₂ (для букв нижнего регистра).

4. Заглавная буква имеет код на 32 (0x20) меньше, чем соответствующая строчная буква.

На базе первого свойства работает двоично-десятичный формат BCD и BCD-арифметика.

Расположение части небуквенных знаков было обусловлено положением их на клавиатурах тех времён (например, Teletype Model 33), куда они перекочевали с механических [пишущих машин](#), в частности "#\$%_&'()). Окончательно отличное от таблицы ASCII положение этих знаков было закреплено в [IBM PC](#).

На первой лекции мы рассматривали специальные символы, такие, как перевод строки, табуляция, ниже приведены ASCII-коды этих символов.

Символ	Код HEX	Расшифровка	Перевод	Описание
EOT	04	end of transmission	конец передачи	Символ используется эмуляторами терминалов в значении « Конец файла »
BEL	07	bell	звуковой сигнал : звонок	Символ часто обозначается как «\a» и используется для подачи звукового сигнала.
TAB	09	tabulation	горизонтальная табуляция	Обозначается как «\t».
LF	0A	line feed	перевод строки	для « UNIX » — одиночный символ « LF »; для « Windows » — последовательность символов « CR LF », обозначается как «\n».
CR	0D	carriage return	возврат каретки	Символ « СК » обозначается как «\r».

Функции ввода-вывода *getchar()* и *putchar()*

Рассмотрим пример программы ввода текста с клавиатуры и печать его на экране. Признаком конца текста в данном случае служит символ точка — “.”

<pre>#include <stdio.h> int main() { char c; while((c = getchar()) != '.') putchar(c); return 0; }</pre>	<pre>Hello <Enter> Hello world. !!!<Enter> world</pre>
---	--

Как правило функции `getchar()` и `putchar()` определяются посредством макросов препроцессора.

Обратите внимание, что несмотря на то что ввод/вывод посимвольный из-за буферизованного ввода данные анализа цикла поступают только после ввода всей строки целиком, т.е. до ввода `<Enter>`. Т.е. если мы введем строку `12132.12312<Enter>` то ниже отобразится строка “12132” до ввода точки.

В программе ниже данная задача реализована в цикле с постусловием.

<pre>#include <stdio.h> int main () { char character; puts("Enter the symbol, '.'is exit:"); do { character = getchar(); // считать введенный со стандартного // потока ввода символ putchar (character); // вывести этот символ } while (character != '.'); // пока введенный символ не точка return 0; }</pre>
--

Разница между программами в том, что в цикле с постусловием точка отобразится дважды, т.к. будет как минимум одна итерация цикла.

Задачи

1. Считать строку и заменить в ней все строчные английские буквы на заглавные.

<pre>#include <stdio.h> int main() { char c; while((c=getchar())!='\n') // спец символ новой строки if(c>='a' && c<='z') // все символы лежат подряд 'a'=97, 'b'=98, 'c'=99, ...</pre>

```

        putchar('A' + (c-'a'));
    else
        putchar(c);
    return 0;
}

```

Здесь используется четвертое свойство таблицы Заглавная буква имеет код на 32 (0x20) меньше, чем соответствующая строчная буква

$'a' - 'A' = 0x61 - 0x41 = 'b' - 'B' = 0x62 - 0x42 \dots = 'z' - 'Z' = 0x7A - 0x5A = 0x20$

2. Дана строка состоящая из английских букв и цифр. Посчитать сумму цифр в ней.

```

#include <stdio.h>
int main()
{
    char c;
    int sum=0;
    while( (c=getchar())!='\n') //спец символ новой строки
        if(c>='0' && c<='9')
            sum+=c-'0';
    printf("%x",sum);
    return 0;
}

```

3. Дана строка состоящая из английских букв и цифр. Составить из цифр в данной строке одно число и поместить его в целочисленную переменную.

```

#include <stdio.h>
int main()
{
    char c;
    int Number=0;
    while( (c=getchar())!='\n') //спец символ новой строки
        if(c>='0' && c<='9')
            Number = Number*10+c-'0';
    printf("%d",Number);
    return 0;
}

```

4. Дана строка состоящая из английских букв и пробелов, в конце строки символ точка. Необходимо удалить из нее все лишние пробелы.

Функции не буферизированного ввода `getch()` и `getche()`

Для небуферизованного (прямого) ввода существуют функции `getch()` и `getche()`.

Для использования этих функций `int getch(void)` и `int getche(void)` нужно подключить заголовочный файл `<conio.h>`.

Функция `getch()` ожидает нажатия клавиши, после которого она сразу же возвращает значение. При этом, символ, введенный с клавиатуры, на экране не отображается. Функция `getche()` работает аналогично, функции `getch()`, но символ, на экране отображается, это называется “локальное эхо”.

При запуске программы мы видим, что символы отображаются сразу, а не после нажатия на <Enter>.

```
#include <stdio.h>
#include <conio.h>
int main ()
{
    char character;
    puts("Exit('.')");
    do
    {
        character = getch(); // считать введенный со стандартного
        потока ввода символ
        if(character>='a' && character<='z') //все символы лежат
        подряд 'a'=97, 'b'=98, 'c'=99, ...
            putchar('A' + (character-'a'));
        else
            putchar(character);
    } while (character != '.'); // пока введенный символ не точка
    return 0;
}
```



Оформление функций

Операторные скобки с которых начинается и заканчивается тело функции всегда располагаются на отдельной строке на том же уровне, что и заголовок, как в примере 1.

Пример 1	Пример 2
<pre>float cube(float a) { return a*a*a;</pre>	<pre>float cube(float a) { return a*a*a;</pre>

Пример 1	Пример 2
<pre>}</pre> 	<pre>}</pre> 

В примере 2 смещены операторные скобки и само тело функции.

Пример 3	Пример 4
<pre>float cube(float a){ return a*a*a; }</pre> 	<pre>float cube(float a) { return a*a*a; }</pre> 

В примере 3 и 4 операторные скобки располагаются не на отдельных строках.

Вычисление корней квадратного уравнения

Дорабатываем систему меню

Давайте сделаем систему меню на основе не буферизированного ввода.

```
#include <stdio.h>
#include <conio.h>
#include <locale.h>
int main(int argc, char **argv)
{
    char Choice;
    setlocale(LC_ALL, "Rus");
    while(1)
    {
        printf("1. Вычисление корней квадратного уравнения\n");
        printf("0. Выход\n");
        printf("Для выход нажмите Q\n");
        Choice = getch();
        switch(Choice)
        {
            case '1':
                SquareEquation();
                break;
            case '0':
            case 'q':
            case 'Q':
```

```

        return 0;
    break;
    default:
        printf("Непонятный выбор %x\n",Choice);
    break;
}
}
return 0;
}

```

Добавляем функции

Сделаем вычисление корней квадратного уравнения отдельной функцией.

```

void SquareEquation(void)
{
    float a,b,c;
    float B,d;
    float X1,X2;
    printf("Вычисление корней квадратного уравнения \\
    \"a*x*x+b*x+c=0\\\"\\n");
    printf("Введите a:\\n");
    scanf ("%f", &a); //1
    printf("Введите b:\\n");
    scanf ("%f", &b); //18
    printf("Введите c:\\n");
    scanf ("%f", &c); //32
    B = b/2;
    if(a!=0)
    {
        d = B*B-a*c;
        if(d<0)
        {
            printf("Корни квадратного уравнения комплексные \\n");
        }
        else
        {
            printf("Корни квадратного уравнения \\n");
            d = sqrtf(d);
            X1 = (-B + d)/a; //2
            printf("X1 = %f \\n",X1);
            X2 = (-B - d)/a; //16
            printf("X2 = %f \\n",X2);
        }
    }
    else
    {
        if(b!=0)
        {
            X1 = -c/b;
            printf("Корень линейного уравнения %f\\n",X1);
        }
    }
}

```

```

    }
    else
    {
        printf("Корней НЕТ!\n");
    }
}

```

Рефакторинг

Как видно выше код получился громоздким и избыточным. Произведем его рефакторинг.

Рефакторинг — это процесс изменения кода, призванный упростить его обслуживание, понимание и расширение, при этом не изменяя его поведение.

Давайте выделим ввод данных и вычисление корней уравнения в отдельные функции *InputFloat* и *CalcRealRoots*. Тогда в *SquareEquation* останется только логика работы программы.

```

float InputFloat(char* message)
{
    float number;
    printf(message);
    scanf("%f",&number);
    return number;
}

void CalcRealRoots(float sqrD,float B,float a)
{
    float X1,X2;
    printf("Корни квадратного уравнения \n");
    float d = sqrtf(sqrD);
    X1 = (-B + d)/a; //2
    printf("X1 = %f \n",X1);
    X2 = (-B - d)/a; //16
    printf("X2 = %f \n",X2);
}

```

Пример применения static-переменных для диагностики

Добавляем диагностику с помощью переменных с модификатором `static`

Давайте сделаем счетчик вызовов в функции *InputFloat* и будем его выводить перед нашим сообщением. Это можно сделать объявив глобальную переменную, но как мы уже знаем это не очень хорошо. Гораздо лучше добавить модификатор `static` для локальной переменной. Таким образом, мы ограничим область видимости переменной функцией *InputFloat*. Это защищает данные от несанкционированного изменения в других местах кода и реализует принцип *инкапсуляции* данных.

```

float InputFloat(char* message)
{
float number;
static int counter = 0;
    counter++;
    printf("%d,%s",counter,message);
    scanf("%f",&number);
    return number;
}

```

Конечный вариант программы

```

#include <stdio.h>
#include <conio.h>
#include <locale.h>
#include <math.h>

float InputFloat(char* message)
{
float number;
static int counter = 0;
    counter++;
    printf("%d,%s",counter,message);
    scanf("%f",&number);
    return number;
}

void CalcRealRoots(float sqrD,float B,float a)
{
float X1,X2;
    printf("Корни квадратного уравнения \n");
    float d = sqrtf(sqrD);
    X1 = (-B + d)/a; //2
    printf("X1 = %f \n",X1);
    X2 = (-B - d)/a; //16
    printf("X2 = %f \n",X2);
}

//Сделаем вычисление корней квадратного уравнения отдельной
//функцией.
void SquareEquation(void)
{
    printf("Вычисление корней квадратного уравнения \\
\\a*x*x+b*x+c=0\\\"\\n");
    float a = InputFloat("Введите a:\n");//1
    float b = InputFloat("Введите b:\n");//18
    float c = InputFloat("Введите c:\n");//32
    float B = b/2;
    if(a!=0)
    {
        float d = B*B-a*c;

```

```

        if(d<0)
        {
            printf("Корни квадратного уравнения комплексные \n");
        }
        else
        {
            CalcRealRoots(d,B,a);
        }
    }
else
{
    if(b!=0)
    {
        float X1 = -c/b;
        printf("Корень линейного уравнения %f\n",X1);
    }
    else
    {
        printf("Корней НЕТ!\n");
    }
}
}
int main(int argc, char **argv)
{
    char Choice;
    setlocale(LC_ALL, "Rus");
    while(1)
    {
        printf("1. Вычисление корней квадратного уравнения\n");
        printf("0. Выход\n");
        printf("Для выход нажмите Q\n");
        Choice = getch();
        switch(Choice)
        {
            case '1':
                SquareEquation();
                break;
            case '0':
            case 'q':
            case 'Q':
                return 0;
                break;
            default:
                printf("Непонятный выбор %x\n",Choice);
                break;
        }
    }
    return 0;
}

```

Машины состояний и конечные автоматы

Конечный автомат (КА) — математическая абстракция, модель дискретного устройства, имеющего один вход, один выход и в каждый момент времени находящегося в одном состоянии из множества возможных.

Например, собака всегда спит или бодрствует. Собака не может спать и бодрствовать одновременно, и собака не может ни спать, ни бодрствовать. Есть только эти два состояния, строго ограниченное, конечное число состояний. Также есть события переводящее собаку из одного состояния в другое.

Эта модель вычислений, основанная на гипотетической машине состояний, которая для выполнения каких-либо действий должна менять свое состояние.

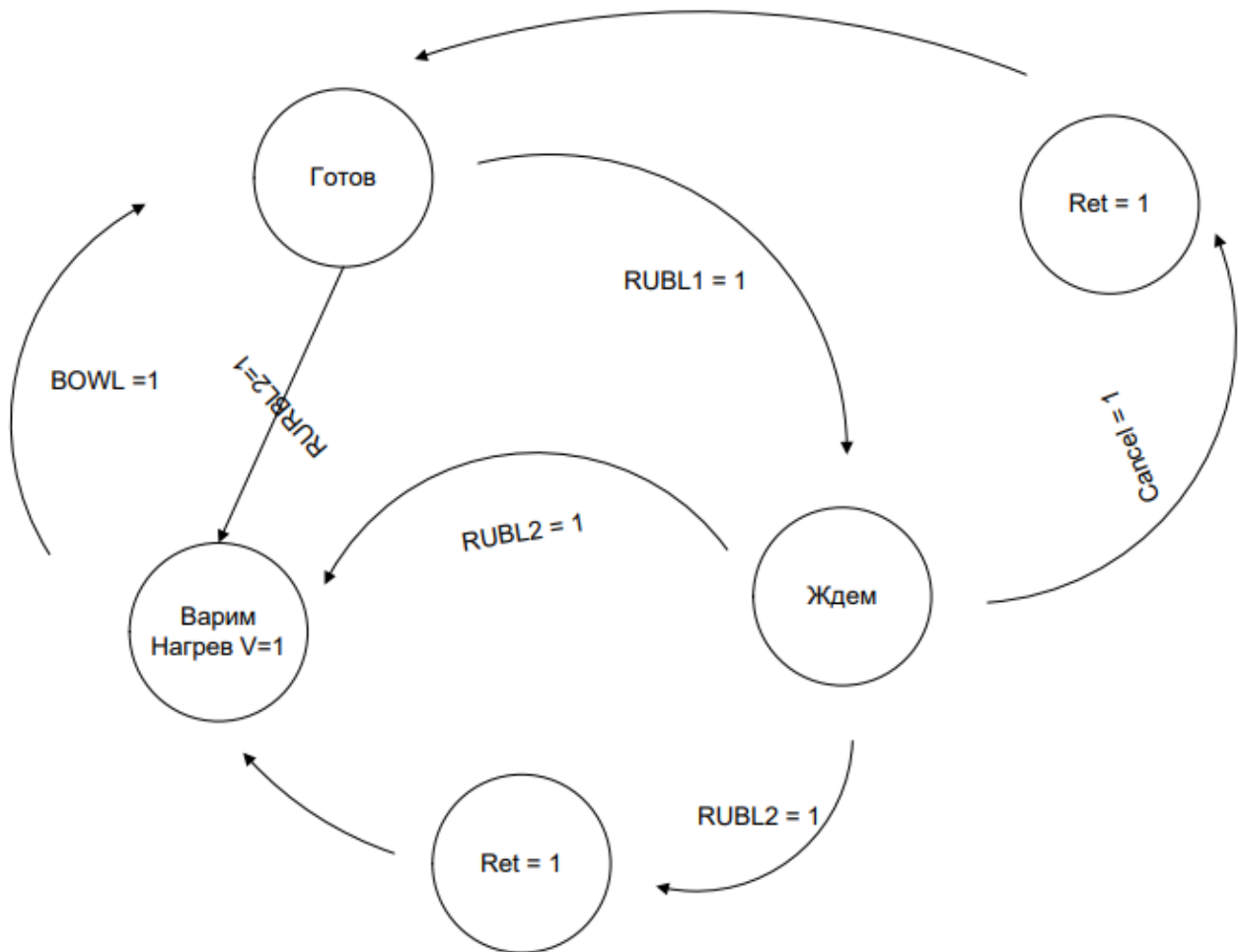
Конечный автомат для варки кофе

Давайте рассмотрим более сложный пример. Автомат для варки кофе работает по следующим правилам:

- он может принимать монеты 1 и 2 рубля
- кофе стоит 2 рубля
- автомат может выдавать сдачу
- автомат может варить кофе
- есть кнопка отмены заказа

Конечный автомат для варки кофе

Принимает монеты 1 и 2 руб., есть кнопка отмены, выдает кофе и сдачу



Давайте определим состояния:

- готов к работе (READY)
- варит кофе (PREPARE)
- возврат сдачи (CHANGE)
- ожидание доплаты (WAIT)
- отмена (RETURN)

Теперь определим события перехода из одного состояния в другое:

- заплатили 1 рубль (Rubl_1)
- заплатили 2 рубля (Rubl_2)
- нажали кнопку отмены заказа (Cancel)

События для перехода мы получаем в функции *GetUserSignal*.

Переключение состояний происходит в функции *main*.

```

#include <stdio.h>
#include <conio.h>

int Rubl_1 = 0, Rubl_2 = 0, Cancel = 0;

void GetUserSignal(void)
{
    char Choice;
    while(1)
    {
        printf("1.Put 1 rubl\n2.Put 2 rubl\n0.Cancel\n");
        Choice = getch();
        switch(Choice)
        {
            case '1': Rubl_1=1; return;
            case '2': Rubl_2=1; return;
            case '0': Cancel=1; return;
        }
    }
}

int main()
{
    enum states { READY, PREPFRE, WAIT, CHANGE, RETURN } state =
READY;
    while (1)
    {
        switch (state)
        {
            case READY:
                printf("Ready\n");
                GetUserSignal();
                if(Rubl_2) state = PREPFRE;
                if(Rubl_1) state = WAIT;
                break;
            case PREPFRE:
                printf("Preapare cofee\n");
                state = READY;
                break;
            case WAIT:
                printf("Wait\n");
                GetUserSignal();
                if(Rubl_2) state = CHANGE;
                if(Rubl_1) state = PREPFRE;
                if(Cancel) state = RETURN;
                break;
            case CHANGE:
                printf("Change 1 Rubl\n");
                state = PREPFRE;
                break;
            case RETURN:
                printf("Change 1 Rubl\n");

```

```

        state = READY;
        break;
    }
    Rubl_1 = 0; Rubl_2 = 0; Cancel = 0;
}
return 0;
}

```

Рефакторинг программы конечного автомата для варки кофе

Давайте уберем совсем глобальные переменные

```

#include <stdio.h>
#include <conio.h>

enum states { READY, PREPFRE, WAIT, CHANGE, RETURN };
enum signals { ONE_RUBL, TWO_RUBLS, CANCEL, NONE };

enum signals GetUserSignal(void)
{
    char Choice;
    while(1)
    {
        printf("1.Put 1 rubl\n2.Put 2 rubl\n0.Cancel\n");
        Choice = getch();
        switch(Choice)
        {
            case '1': return ONE_RUBL;
            case '2': return TWO_RUBLS;
            case '0': return CANCEL;
        }
    }
}

```

```

int main()
{
    enum states state = READY;
    enum signals signal = NONE;
    while (1)
    {
        switch (state)
        {
            case READY:
                printf("Ready\n");
                signal = GetUserSignal();
                if(signal==TWO_RUBLS)
                    state = PREPFRE;
                if(signal==ONE_RUBL)

```

```

            case WAIT:
                printf("Wait\n");
                signal = GetUserSignal();
                if(signal==TWO_RUBLS)
                    state = CHANGE;
                if(signal==ONE_RUBL)
                    state = PREPFRE;
                if(signal==CANCEL)
                    state = RETURN;
                break;
            case CHANGE:
                printf("Change 1 Rubl\n");
                state = PREPFRE;
                break;

```

<pre> state = WAIT; break; case PREPFRE: printf("Preapare cofee\n"); state = READY; break; </pre>	<pre> case RETURN: printf("Change 1 Rubl\n"); state = READY; break; } } return 0; } </pre>
---	--

Получился очень изящный код

Подведение итогов

Итак, на этой лекции мы начали с изучения [функций](#) их [определение и вызовов](#). Зачем нужен [прототип или описание функции](#), как передавать [параметры в функцию и получать результат](#). Рассмотрели вопросы [область видимости](#) и модификатор static.

Изучили особенности [буферного и не буферного ввода](#) и как работают [потoki ввода и вывода](#).

Познакомились с [ASCII-таблицей](#) и ее свойствами.

Рассмотрели [функции ввода-вывода getchar\(\) и putchar\(\)](#), а также [функции не буферизированного ввода getch\(\) и getche\(\)](#).

Затронули тему [оформление функций](#).

Традиционно продолжили написание программы [вычисления корней квадратного уравнения](#) в части [доработки систему меню](#) и [добавляя функций](#), а также произвели [рефакторинг](#), кода. Еще раз на практике рассмотрели [модификатор static переменных](#).

Познакомились с [машиной состояний и конечными автоматами](#) и реализовали [конечный автомат машины для варки кофе](#).

Дополнительные материалы

1. Оформление программного кода
<http://www.stolyarov.info/books/pdf/codestyle2.pdf>
2. Стандарт разработки программного обеспечения MISRA на языке C
http://easyelectronics.ru/files/Book/misra_c_rus.pdf

3. <https://ru.wikipedia.org/wiki/ASCII#%D0%9D%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5%D0%B2%D0%B0%D1%80%D0%B8%D0%B0%D0%BD%D1%82%D1%8B> ASCII ASCII
4. <https://habr.com/ru/company/ruvds/blog/548672/> Решайтесь на великие поступки — ASCII
5. <https://blog.skillfactory.ru/glossary/ascii/> ASCII
6. <https://www.ap-impulse.com/shag-30-ds1307-i-avr-dvoichno-desyatchnyj-format-bcd-vyvodim-vremya-na-indikator/> Шаг №30. DS1307 и AVR. Двоично-десятичный формат BCD
7. <https://habr.com/ru/companies/timeweb/articles/717628/> С чем едят конечный автомат
8. <https://tproger.ru/translations/finite-state-machines-theory-and-implementation/> Конечный автомат: теория и реализация
9. <https://metanit.com/c/tutorial/7.7.php> Консольный ввод-вывод
10. https://cpp.com.ru/kr_cbook/ch7kr.html Глава 7. Ввод и вывод

Используемые источники

1. cprogramming.com - учебники по [C](#) и [C++](#)
2. free-programming-books - ресурс содержащий множество книг и статей по программированию, в том числе по [C](#) и [C++](#) (там же можно найти ссылку на распространяемую бесплатно автором html-версию книги Eckel B. «Thinking in C++»)
3. tutorialspoint.com - ещё один ресурс с множеством руководств по изучению различных языков и технологий, в том числе содержит учебники по [C](#)
4. Юричев Д. [«Заметки о языке программирования Си/Си++»](#) - «для тех, кто хочет освежить свои знания по Си/Си++»
5. [Онлайн версия «Си для встраиваемых систем»](#) Функции
6. <https://metanit.com/c/tutorial/4.1.php> Функции