Структура языка Си
Основные функции ввода и вывода
Операторы языка
Обзор циклических операторов

Язык программирования C/C++

Преподаватель: к.т.н., доцент Женса Андрей Вячеславович

Ведущий программист: Варданян Андраник Эдуардович

Ведущий программист: Лебедев Данила Александрович

Аспирант: Бабкин Михаил Андреевич

Язык Си

Язык Си — это стандартизированный процедурный язык программирования, разработанный в 1972 году сотрудником компании AT&T Bell Laboratories Деннисом Ритчи (англ. Dennis Ritchie) в качестве развития языка Би

Особенности языка

- небольшое количество ключевых слов
- наличие сложных типов данных (структуры, объединения)
- возможность использования указателей, работа с памятью
- наличие внешних стандартных библиотек
- компиляция программного кода в бинарный код
- использование макропроцессора

Сравнение с другими языками

- язык является прародителем C++, Objective C, C#
- язык сильно повлиял на Java, Perl, Python
- низкий уровень языка Си предоставляет выигрыш в скорости исполнения кода
- недостатки по сравнению с другими языками:
  - отсутствие исключений (exceptions)
  - отсутствие проверки диапазонов (range-checking)
  - отсутствие автоматической сборки мусора
  - Си не является объектно-ориентированным языком
  - отсутствие полиморфизма

Алфавит языка Си

В алфавит входят:

- прописные и строчные буквы латинского алфавита A-Z, a-z;
- цифры 0-9;
- специальные знаки: . , ; : ? ! " „ + - * / % ( ) [ ] { } < = > \ & # _ ~ ^;
- символы пробела: пробел, табуляция, символ конца строки.

Таблица 1. Наименования символов языка Си.

Символ	Наименование	Символ	Наименование
,	запятая	+	плюс
.	точка	-	минус (дефис)
;	точка с запятой	( )	круглые скобки
:	двоеточие	{ }	фигурные скобки
?	вопросительный знак	[ ]	квадратные скобки
!	восклицательный знак	<	меньше
'	одинарная кавычка (апостроф)	>	больше
|	вертикальная черта	#	решетка
/	дробная черта (слеш)	%	процент
\	обратная черта (обратный слеш)	&	амперсанд

Типы данных:

- Символьный тип данных (char)
- Числовые типы данных (int, double, float)
- Логический тип данных (_Bool)
- Тип без значения (void)

- **Символьный тип данных (char)** содержит данные в виде единичных символов, букв, цифр или знаков. При написании кода программы единичные символы заключают в одинарные кавычки.
- **Числовые типы данных (int, float, double)** включают в себя целые и дробные числа.
  - int – целые числа
  - float – дробные числа (одинарная точность 4 знака после запятой)
  - double – дробные числа (двойная точность 8 знаков после запятой)
- **Логический тип данных (_Bool)** позволяет хранить значения:
  - 1 (true)
  - 0 (false)
- **Тип без** значения (void) служит для объявления функций, которые не возвращают значения, и создания универсальных указателей.

Модификации простых базовых типов.

Спецификаторы длины и спецификаторы знака: **signed, unsigned, short, long**

Пример:

unsigned int a;

long double b;

Сложные типы данных.

На основе базовых типов можно образовать сколь угодно сложные и многоуровневые типы данных:

- массивы;
- структуры;
- перечисления;
- объединения;
- битовые поля.

Хранение данных

- **Переменная** — это идентификатор, скрывающий за собой область памяти с хранящимися там данными. Иначе говоря, это имя области памяти;
- Переменная имеет тип, который соответствует тому, какой тип данных она хранит;
- *Имя переменной может состоять из записанных в любом порядке символов латинского алфавита, цифр и символа подчеркивания. При этом первым символом имени переменной не может быть цифра.*

Пример:

```
int sum;
double average;
```

Прим: имя переменной должна быть логичной, чтобы программист читающий код, знал какую информацию она в себе хранит!

Объявление переменных

- Объявление переменных может быть расположено в трех местах программы: внутри функции, вне всех функций и при определении параметров функции.

тип_данных имя_переменной [= начальное_значение];

Пример:

```
int sum; // объявление одной переменной
double average, sum; // обявление нескольких переменных одного типа
double sum = 100; // объявление с инициализацией
```

Константы

- Константа — это фиксированное значение, которое не может быть изменено программой. Она может относиться к любому базовому типу.

Константы выполняют несколько важных функций:

- использование констант как размерности массивов (размерность массива не может быть переменной). (Этот способ употребления констант может не работать при использовании некоторых версий компилятора.)
- слежение за тем, чтобы случайным образом не изменить величину какой- либо переменной
- для удобства пользователя и разграничения числовых значений в программе

const тип_константы имя_константы = значение;

const int mount = 10;

Директива #define

- Директива #define определяет идентификатор и последовательность символов, которая во время компиляции программы будет подставляться вместо идентификатора каждый раз, когда он встретится. Идентификатор называется макросом, а сам процесс замены — макрозаменой. В общем виде директива выглядит таким образом:

#define имя_макроса последовательность_символов

Пример:

#define M 10

Функция консольного вывода printf()

- Функция printf() выводит информацию на экран с учетом выбранного форматирования. На вход функции подается строка форматирования и список параметров.

printf(" строка форматирования ", список параметров);

Строка форматирования — это специальная последовательность символов, которая отображает, как именно вы хотите записать число или символ. Рассмотрим ее более подробно.

"%[флаг][ширина][.точность]тип"

Типы аргумента в строке форматирования

- d (decimal, указывает на int);
- f (float, указывает на float);
- lf (long float, указывает на double);
- c (char, указывает на char);
- s (string, указывает на строку);

Примеры вывода

Вывод целых чисел

1	printf ("%d", 3);		3							
2	printf ("%4d", 3);					3				
3	printf ("%-4d", 3);		3							
4	printf ("%+7d", 3);							+	3	
5	printf ("%+-3d", 3);		+	3						
6	printf ("%0+5d", 3);		+	0	0	0	3			
7	printf ("%0+-5d", 3);		+	3						
8	printf ("%2d", 3675);		3	6	7	5				

Вывод дробных чисел

1	printf ("%lf", 2.157);		2	.	1	5	7	0	0	0	
2	printf ("%5.1lf", 2.157);				2	.	2				
3	printf ("%5lf", 2.157);		2	.	1	5	7	0	0	0	
4	printf ("%.1lf", 2.157);		2	.	2						
5	printf ("%7.5lf",2.157);		2	.	1	5	7	0	0		
6	printf ("%+6.2lf", 2.157);			+	2	.	1	6			
7	printf ("%-7.0lf", 2.157);		2								
8	printf ("%+07.2lf", 2.157);		+	0	0	2	.	1	6		
9	printf ("%+-07.2lf", 2.157);		+	2	.	1	6				

Вывод нескольких значений разных типов. Сложное форматирование.

- Если вы хотите вывести не одно, а сразу несколько значений, то это нужно отразить и в строке форматирования, и в списке параметров. Например

printf("%d %lf %d %c", 45, 95.1, -72, 'f');

выведет нам в строку через пробел:

45 95.100000 -72 f

Если же мы не поставим пробелы в строке форматирования

printf("%d%lf%d%c", 45, 95.1, -72, 'f');

то увидим:

4595.100000-72f

Если вы не ставите пробелы в строке форматирования, то и на экране их тоже не будет.

Экранирование

Экранирование (обратный слеш)	
управляющие символы	**значение**
\n	новая строка
\t	табуляция
\'	одинарная кавычка
\"	двойная кавычка
\\	обратный слеш
%%	знак процента

Примеры

- printf("Строка1\nСтрока2\nСтрока3");
  Строка1

  Строка2

  Строка3
- printf("Это обычная строка\n\tА это красная строка");

  Это обычная строка

  А это красная строка

Функция ввода с клавиатуры scanf()

Функция scanf() считывает с клавиатуры символы, строки, числа и записывает их в адрес указанных переменных. Синтаксис функции очень похож и даже практически идентичен функции printf()

scanf("строка форматирования", адреса переменных);

Строка scanf("%d", &x); означает, что мы считываем с клавиатуры целое число и записываем его в адрес (&) переменной x. Фактически переменная x станет равной числу, которое мы введем с клавиатуры

Примеры

- scanf("%lf", &y); *// дробное значение для переменной y*
- scanf("%d %lf", &a, &b); *// целое a и дробное b через пробел*
- scanf("%c %c", &str1, &str2); *// два символа через пробел*
- scanf("%d.%d.%d", &day, &month, &year); *// печатаем «15.05.2023»*
- scanf("мимо пробегали %d розовых слоников", &elephants); *//здесь тоже печатаем фразу целиком: «мимо пробегали 8 розовых слоников».* ***Проще говоря, как написано в строке форматирования, так и нужно вводить данные с клавиатуры.***

Одиночный оператор.

Любое выражение, которое заканчивается точкой с запятой, является оператором.

- func(); // вызов функции
- a = b + c; // оператор присваивания
- b++; // оператор инкремента
- ; // пустой оператор

Первый оператор выполняет вызов функции, второй присваивание. Третий оператор увеличивает значение переменной на единицу. Последний оператор – это пустой оператор, который не выполняет никаких действий.

Блок операторов

- Блок операторов — это последовательность операторов, заключенных в фигурные скобки, которые рассматриваются как одна программная единица. Операторы, составляющие блок, логически связаны друг с другом. Блок всегда начинается открывающейся фигурной скобкой { и заканчивается закрывающейся }. Чаще всего блок используется как составная часть какого-либо оператора, выполняющего действие над группой операторов, например, if или for. Однако блок можно поставить в любом месте, где может находиться оператор.

```
int x = 0, y = 5, z = -3;
{// это блок операторов
    x = y + z;
    printf("результат сложения равен %d", x);
}
for (x = 0; x < 5; x += 2) {
// это тоже блок операторов
    printf("четное число %d\t", x);
    printf ("нечетное число %d\n", x + 1);
}
```

Оператор присваивания.

- Оператор присваивания может присутствовать в любом выражении языка Си. Этим Си отличается от большинства других языков программирования, в которых присваивание возможно только в отдельном операторе.

  Общая форма оператора присваивания:

  имя_переменной = выражение;

  Выражение может быть просто константой или сколь угодно сложным выражением. Оператором присваивания служит знак "=". Адресатом (получателем, левой частью оператора) присваивания должен быть объект, способный получить значение, например переменная.

Множественные присваивания

- В одном операторе присваивания можно присвоить одинаковое значение многим переменным. Для этого используется оператор множественного присваивания:

  x = y = z = 0;

При этом операции присваивания выполняются справа налево. Таким образом, сначала выполнится операция z = 0, затем y = z, и, наконец, x = y.

Составное присваивание

- Составное присваивание — это разновидность оператора присваивания, в которой запись сокращается и становится более удобной в написании. Например, оператор *a = a + 35*; можно записать как *a* += 35;
- Оператор "+=" сообщает компилятору, что к переменной *a* нужно прибавить *35*. Составные операторы присваивания существуют для всех бинарных операций.

Составные операторы присваивания.

Обычный оператор присваивания	Составной оператор присваивания
x = x + 3;	x += 3;
x = x - 3;	x -= 3;
x = x * 3;	x *= 3;
x = x / 3;	x /= 3;
x = x % 3;	x %= 3;

Арифметические операции на переменными

Оператор	Операция	Класс операций	
++	инкремент (a++)	аддитивные	унарные
--	декремент (a--)		
+	сложение (a + b)		бинарные
-	вычитание (a - b)		
-	унарный минус (- a)	мультипликативные	унарная
*	умножение (a * b)		бинарные
/	деление (a / b)		
%	остаток от деления (a % b)		

Арифметические операции

- Унарные арифметические операции производятся с использованием только одной переменной.
- Операции называются бинарными, если в них участвуют две переменные
- Аддитивные арифметические операции — это операции сложения и вычитания.
- Мультипликативные операции — это операции умножения, деления и нахождения остатка от деления целого числа на целое число.

Стандартная библиотека математических функций math.h

Математический вид	Библиотека math.h	Описание функции
$\|x\|$	abs(x)	модуль целого числа
$\|x\|$	fabs(x)	модуль дробного числа
$\sqrt{x}$	sqrt(x)	квадратный корень
a^x	pow(a, x)	возведение в степень
e^x	exp(x)	экспонента
ln(x)	log(x)	натуральный логарифм
lg(x)	log10(x)	десятичный логарифм
sin(x)	sin(x)	синус
cos(x)	cos(x)	косинус
tan(x)	tan(x)	тангенс
--	round(x)	округление до целого
--	ceil(x)	округление до большего целого
--	floor(x)	округление до меньшего целого

Стандартная библиотека stdlib.h Генерация псевдослучайных чисел

- Псевдослучайные числа (далее - случайные числа) генерируются с помощью функции rand(). Эта функция возвращает целое число в диапазоне от нуля до RAND_MAX. Случайное число генерируется алгоритмом, который возвращает последовательность внешне не связанных цифр при каждом вызове. Этот алгоритм использует серии вызовов, которые должны быть инициализированы значением с помощью функции srand().
- RAND_MAX - это константа, определенная в библиотеке . Ее значение может варьироваться в зависимости от используемой реализации языка Си, но она не меньше, чем 32767.

Генерация псевдослучайных чисел

- int x = rand() % 10; //от 0 до 9
- int x = -328 + rand() % 1814; //от -328 до 1485
- int x = -328 + rand() % (1485 - (-328) +1);

Инициализация генератора случайных чисел

```
#include <time.h>
…
srand(time(0));
```

Явное и неявное преобразование типов

- Неявное преобразование типов данных выполняет компилятор, а явное преобразование данных выполняет сам программист.

int x = 5, y = 2; int z = x / y;

здесь z = 2, так как это целочисленное деление, и вся дробная часть отбрасывается.

int x = 5, y = 2; double z = x / y;

(z = 2.0) здесь z = 2.0, так как сначала происходит целочисленное деление, а затем преобразование целого получившегося числа 2 в дробное число 2.0 для записи в дробную переменную z. Целое число делим на дробное:

int x = 5; double y = 2.0; double z = x / y; //(z = 2.5)

Дробное число делим на целое:

int y = 2; double x = 5.0; double z = x / y; //(z = 2.5)

Неявные преобразования

- **double f = 50;** здесь дробной переменной f присваивается значение целого числа 50. При этом число 50 преобразуется компилятором языка в дробное число 50.0, а затем происходит операция присваивания.
- **int var1 = -34; double var2 = var1;** Здесь целое значение переменной var1 преобразуется компилятором в дробное значение -34.0 (при этом переменная var1 остается целого типа, и в ней ничего не меняется). А затем происходит присвоение переменной var2 дробного значения -34.0.
- **double age = 9.8; int class = age - 7;** Если целой переменной присвоить дробное значение, то произойдет отбрасывание дробной части. Здесь сначала выполнится вычитание age — 7. Результат такого выражения будет дробным 2.8. Перед присвоением отбрасывается дробная часть, и получается целое число 2, которое присваивается переменной class.

Явное преобразование типов

- Для того, чтобы произвести явные преобразования типа переменной или результата вычислений, используются операции преобразования типа:

(тип, к которому нужно преобразовать) выражение

```
double x;
x = (double)5;
```

Операции сравнения

- Операции сравнения — это операции, в которых значения двух переменных или выражений сравниваются друг с другом. Логические же операции реализуют средствами языка Си операции формальной логики. Между логическими операциями и операциями сравнения существует тесная связь: результаты операций сравнения часто являются операндами логических операций.
- В операциях сравнения и логических операциях в качестве операндов и результатов операций используются значения ИСТИНА (true) и ЛОЖЬ (false). В языке Си значение ИСТИНА представляется любым числом, отличным от нуля. Значение ЛОЖЬ представляется нулем.

Операторы сравнения

Операторы сравнения	
Оператор	**Операция**
>	больше
>=	больше или равно
<	меньше
<=	меньше или равно
==	равно
!=	не равно
Логические операторы	
Оператор	**Операция**
&&	И
||	ИЛИ
!	НЕ

Операторы ветвления

- **Тернарный оператор**

Оператор ?: Называют тернарным оператором (от лат. ternarius — «тройной»).

Общий вид операции:

(Условие) ? Оператор1 : Оператор2;

Если условие выполняется, то исполнятся Оператор1, если нет — то Оператор2.

```
max = (x > y) ? x : y;
printf ("Наибольшее число равно %d", max);
```

Тернарную операцию можно включать в другие операции.

```
printf ("Наибольшее число = %d", (x > y) ? x : y);
```

Здесь переменная max не обязательна. Вычисление наибольшего из x и y происходит прямо в функции printf() , без использования промежуточной переменной.

- Тернарные операции также могут быть вложенными. В качестве примера можно рассмотреть поиск наибольшего из трех различных чисел.

В две строки (без вложенности):

```
max = (a > b) ? a : b; max = (max > c) ? max : c;
```

В одну строку (вложенность):

```
max = (((a > b) ? a : b) > c) ? ((a > b) ? a : b) : c;
```

Оператор if

- Общая (полная) форма оператора if следующая:

```
if(выражение/условие)
    оператор;
else
    оператор;
```

```
if(выражение/условие) {
    блок операторов;
}
else {
    блок операторов;
}
```

Условный оператор может иметь неполную форму, при которой else и последующий оператор (или блок операторов) отсутствуют.

```
if (выражение/условие)
{
 оператор;
```

```
if (выражение/условие)
    блок операторов;
}
```

- **Пример 1.**

Модуль целого числа.

```
int x;
printf("Введите x:");
scanf("%d", &x);
if (x < 0)
 x *= -1;
printf("модуль x = %d\n", x);
```

Если мы ввели с клавиатуры отрицательное число, то для вычисления модуля нужно умножить его на минус единицу. Если же число не отрицательно, то, ни на что умножать не надо, его модуль будет равен самому числу. Поэтому отсутствует часть else.

- Пример 2.

Вычислить значение переменной с условием.

```
 int x = 5, y = 7;
double z;
if ((x * y > 50) || (x / y < 10))
 z = (x + y) * 10.4;
else
 z = (35 * x - 12 * y) * 0.5;
printf("z=%lf", z);
```

Вычисление значения переменной z в зависимости от условий. Переменная вычисляется по одной из двух формул и результат вычисления выводится на экран.

Если у вас не одно, а несколько условий, то каждое из них рекомендуется заключить в круглые скобки. При этом не забывайте об общих круглых скобках.

```
if ((условие1) || (условие2) && (условие3))
```

- Пример 3.

Вычисление ширины квадрата.

```
int square = 621, length = 0, height = 0;
printf("Enter length");
scanf("%d", &length);
if (length > 0) {
 height = square / length;
 printf("height = %d\n", height); }
else
 printf("ERROR: length is wrong!\n");
```

Вычисление ширины квадрата, если известна его площадь, а длина вводится с клавиатуры. Если длина положительная, то мы вычисляем ширину. Иначе мы выдаем сообщение о том, что длина квадрата введена неверно.

- Пример 4. Наибольшее из трех чисел .

```
int a, b, c;
scanf("%d %d %d", &a, &b, &c);
if (a > b)
  if (a > c)
        printf("a – наибольшее число");
  else
        printf("c – наибольшее число");
else
  if (b > c)
        printf("b – наибольшее число");
  else
            printf("c – наибольшее число");
```

Операторами для if и else могут быть так же условные операторы. При этом вложенность может быть многократная.

Условный оператор множественного выбора (переключатель) switch.

```
switch (селектор){
    case значение1:
        блок операторов;
        break;
    case значение2:
        блок операторов;
        break;
    case значение3:
        блок операторов;
        break;
    case значение4:
        блок операторов;
        break;
        default:
        блок операторов;
        break;
    }
```

Селектор — это переменная или выражение, которое должно иметь целочисленный или символьный тип.

Пример использования switch

```
printf("Введите оценку ученика");
scanf("%d", &mark);
switch(mark){
        case 2:
                printf("Неуспевающий ученик");
                break;
        case 3:
                printf("Слабый ученик");
                break;
        case 4:
                printf("Хорошист");
                break;
        case 5:
                printf("Отличник");
                break;
        default:
                printf("Неверная оценка ученика") ;
                break;
}
```

Операторы цикла

- Циклические операции являются часто употребляемыми операциями. Они служат для многократного выполнения последовательности операторов до тех пор, пока не выполниться некоторое условие. Условие может быть установлено заранее или меняться при выполнении тела цикла.

- Цикл while

Общая форма цикла while имеет вид:

```
while (условие) {
 блок операторов;
}
```

Пример: Вывести на экран числа от 0 до 10.

```
int number = 0;
while (number <= 10) {
 printf("%3d", number); // вывод на экран
 number++; // переход к следующему числу
}
```

- Пример: Вычислить через сколько минут переполнится дырявая бочка с водой (250 литров), если каждую минуту (кроме шестой) в нее вливается по 3 литра воды, а каждую шестую минуту единовременно вытекает 5 литров.

```
#define MAX_VOLUME 250
int volume = 0, time = 0;
while (volume < MAX_VOLUME) {
  time++;
  if (time % 6 == 0) {
      volume -= 5;
          }
  else {
      volume += 3;
  }
}
printf("Бочка переполнится через %d минут\n", time);
```

Введем макрос для максимального объема бочки и две переменные для текущего объема воды в бочке и времени. Предварительно обнулим переменные. Далее в цикле будем прибавлять минуты и прибавлять или отнимать литры в зависимости от того, какая идет минута. После выполнения цикла на экран выводится количество минут до переполнения бочки.

- Цикл do-while

Общая форма цикла do-while имеет вид:

```
do {
 блок операторов;
} while();
```

В отличие от цикла while, в котором условие проверяется до выполнения, цикл do-while сначала выполняет тело цикла, а затем проверяет условие. Таким образом, цикл do-while выполняется по крайней мере один раз, а цикл while может не выполниться ни разу.

- Пример: На какой раз выпадет число ноль при бросании случайного числа в пределах [0; 1].

```
long int count = 0;
double number = 0;
do {
 number = rand() / (double)RAND_MAX;
 count++;
}  while (number != 0);
printf("Ноль выпадет на %li раз\n", count);
```

Объявляем одну целую перемененную для подсчета количества бросков и дробную переменную для записи случайного дробного числа от нуля до единицы. В теле цикла do-while будет генерироваться случайное число. После генерации числа счетчик количества генераций увеличивается на единицу и проверяется условие цикла. Если условие выполняется, то цикл повторяется.

Пример: Задать с клавиатуры отрицательное значение для переменной.

цикл while:

```
int number = 0;
printf("Enter a negative number\n");
scanf ( "%d",  &number);
while (number >= 0) {
 printf("Enter a negative number\n");
 scanf( "%d", &number);
}
printf ("The negative number is %d\n", number);
```

- цикл do-while:

```
int number = 0;
do {
 printf("Enter a negative number\n");
 scanf("%d", &number);
} while (number >= 0);
printf("The negative number is %d\n",number);
```

Цикл for.

- Общая форма цикла for имеет вид:

```
for (инициализация; условие; приращение) {
 блок операторов;
}
```

Инициализация — это присваивание начального значения переменной, которая называется параметром цикла.

Условие представляет собой условное выражение, определяющее, следует ли выполнять в очередной раз тело цикла.

Оператор приращения осуществляет изменение параметра цикла при каждой итерации.

Пример: Вывести на экран все целые числа от 0 до 100.

```
int x;
for (x = 0; x <= 100; x++) {
 printf("%4d", x);
}
```

- Пример: Вычислить и вывести на экран координаты точек функции y=7x-5x^2 при значениях x [-15; 30] с шагом 3.

```
int x = 0, y = 0;
for (x = -15; x <= 30; x += 3) {
 y = 7 * x - 5 * x * x;
 printf("x = %3d\ty = %5d\n", x, y);
}
```

Параметр цикла изменяется от -15 до 30. Тело цикла состоит из вычисления значения переменной y и оператора вывода значений переменных x и y на экран в столбик.

Множество параметров цикла for

- В качестве параметров для цикла for может служить несколько переменных. В этом случае они могут указываться через запятую.

Пример: Сколько потребуется итераций для того, чтобы сумма трех переменных стала больше либо равной 100. Начальные значения переменных: 1, 0, 0. Шаг изменения переменных: +1, -1, +7.

```
int x = 0, y = 0, z = 0, iteration = 0;
for (x = 1, y = 0; x + y + z < 100; x++, y--, z += 7) {
 iteration++;
 printf("x = %4d\ty = %4d\tz = %4d\n", x, y, z);
}
printf("\nПрошло %d итераций\n", iteration);
```

Вложенные циклы

- Тело цикла может содержать в себе различные операторы. Это могут быть операторы присваивания, ветвления, циклические операторы и т.д. Примеры циклов, содержащих условные операторы, были представлены выше. Рассмотрим примеры циклов, которые содержат в себе другие циклы.

Пример: Вывести на экран таблицу умножения.

```
    int i, j;
    for (i = 1; i <= 9; i++) {
     for (j = 1; j <= 9; j++) {
          printf("%3d", i * j);
     }
    printf("\n");
}
```

- Пример: Сгенерировать и вывести на экран случайное количество строчек (1..25) из случайного количества (1..50) случайных чисел (-50..50).

```
int num_str = 1 + rand() % 25, num_numb = 0;
int i, j;
for (i = 1; i <= num_str; i++) {
   num_numb = 1 + rand() % 50;
   for (j = 1; j <= num_numb; j++) {
    printf("%4d", -50 + rand() % 101);
   }
printf("\n");
}
```

Бесконечные циклы

- Для создания бесконечных циклов можно использовать любой из трех операторов цикла. Многие программисты чаще всего выбирают цикл while, или цикл for. Самый простой способ создания бесконечного цикла for — это оставить пустыми все три секции.

```
for ( ; ; ) {
  printf("Это бесконечный цикл\n");
}
```

Пример. Найти первое положительное число, прибавляя к числу -389 по 5.

```
int number = -389;
for ( ; ; ) {
  if (number > 0) {
      printf("First \"+\" number is %d\n", number);
      break;
  }
  number += 5;
}
```

Оператор break

Этот оператор применяется в двух случаях:

1. для немедленного выхода из любых циклических операций
2. 2. для прекращения работы оператора условия switch

Пример. Выполнение цикла 100 раз.

```
for (t = 1; t <= 100; t++) {
    count = 1;
    for ( ; ; ) {
        printf("%3d", count);
        count++;
        if (count == 10) {
            break;
        }
    }
}
```

Оператор continue

- Оператор continue прерывает текущую операцию цикла и осуществляет переход к следующей итерации цикла. При этом пропускаются все операторы тела цикла, которые стоят после continue.

Пример. Вывести на экран символы, которые следуют за символами, введенными с клавиатуры. Например, если с клавиатуры ввели цифру «3», то на экран выводится цифра «4», а если ввели букву «m», то выводится буква «n». Ввод производить по одному символу. Если встретится точка, то ввод символов следует прекратить.

```
char ch;
int finish = 0;
while (finish == 0) {
 scanf("%c", &ch);
 if (ch == '.') {
     finish = 1;
     continue;
 }
 printf("%c ", ch + 1);
}
```

Пример таблица умножения

```
int i, j;
for (i = 1; i <= 9; i++){
    for (j = 1; j <= 9; j++) {
        printf("%3d", i * j);
    }
    printf("\n");
}
```

Результат

1 2 3 4 5 6 7 8 9
2 4 6 8 10 12 14 16 18
3 6 9 12 15 18 21 24 27
4 8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81

Спасибо за внимание !