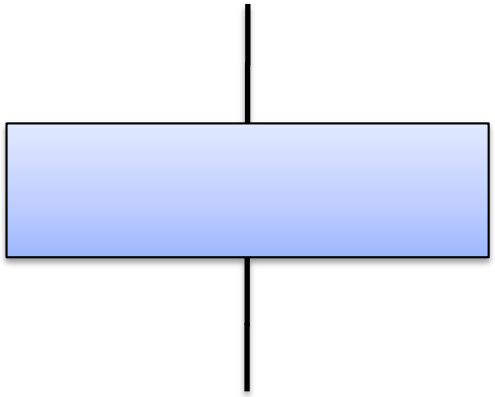
Лекция 6. Управляющие операторы Си: ветвление и переходы.

Ревун Артем Леонидович
ст. преп. кафедры вычислительных систем

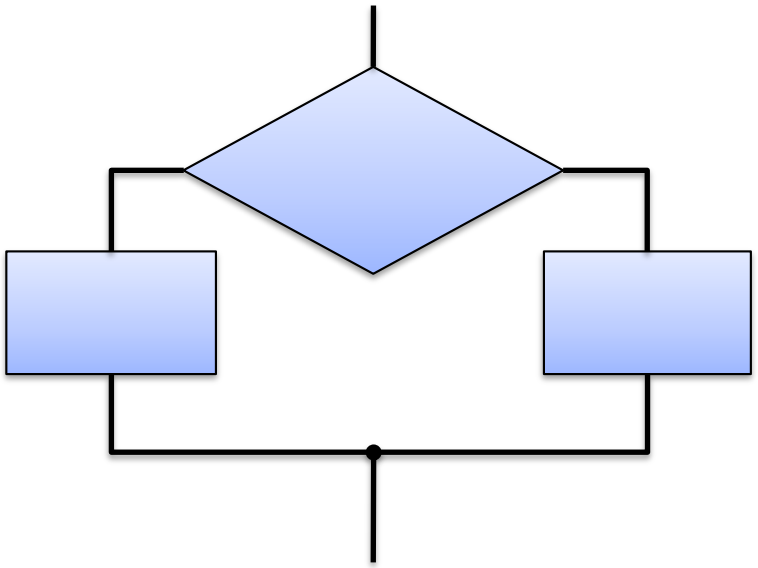
Курс «Программирование», осенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)

Базовые конструкции

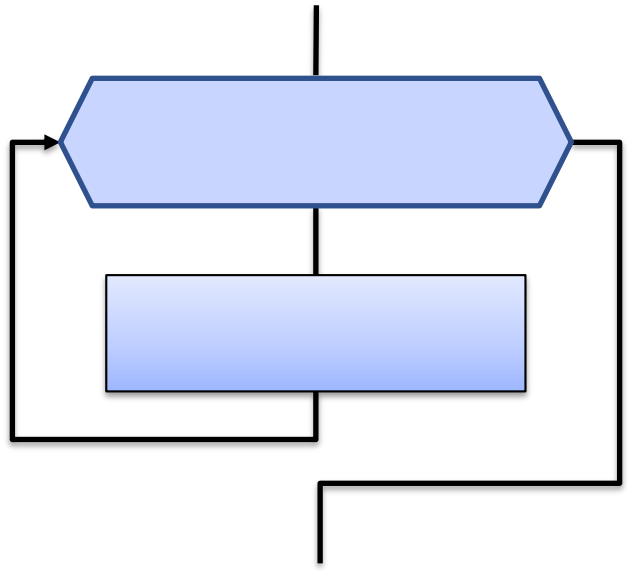


Следование

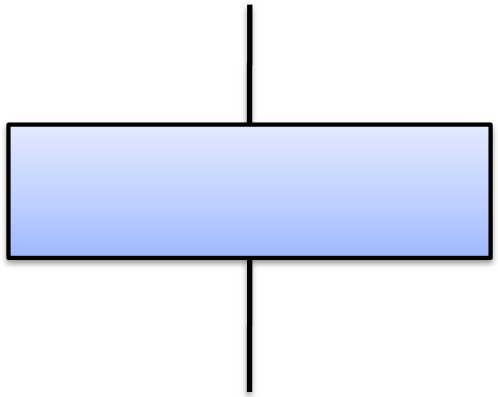
Ветвление



Цикл



Конструкция следование



Следование

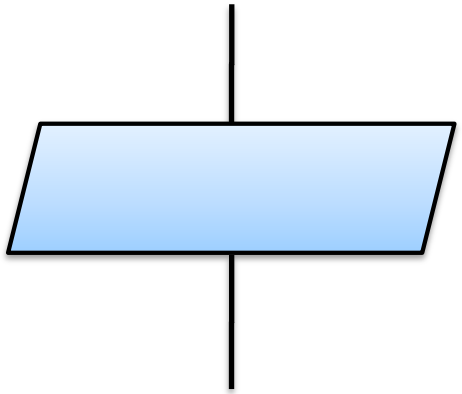
Операторы в языке Си выполняются *последовательно*. После завершения текущего оператора производится переход к следующему оператору по тексту программы.

Исключением являются: операторы, входящие в состав ветвления или цикла.

Например:

```
// Определение переменных
int a = 10, b = 5, c;
// операторы
c = a + b;
c = c * 2;
a = c / b;
b = a * 1.5;
```

Конструкция следование (ввод/вывод)



Следование (ввод/вывод)

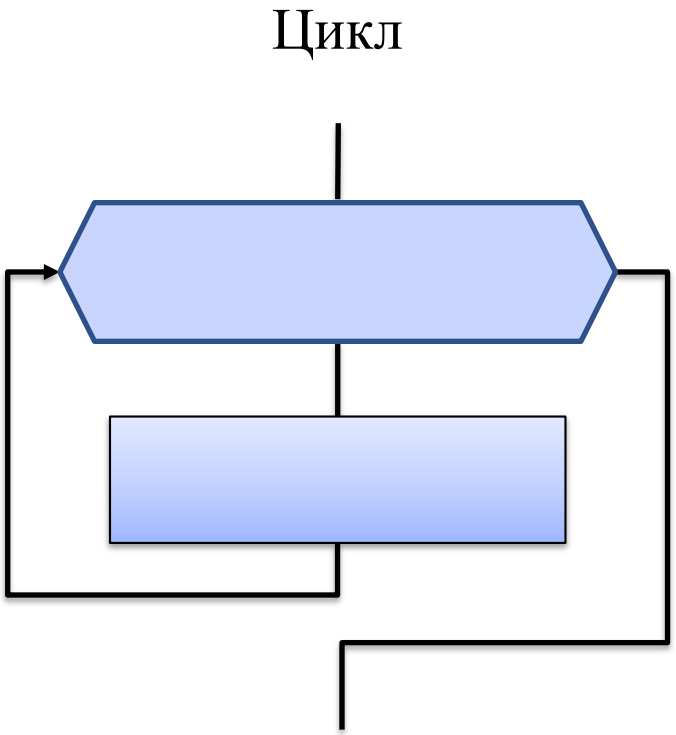
Для обозначения операций ввода/вывода принято использовать параллелограмм.

Внутри указывается название операции **ввод** или **вывод** и название переменной.

Большой текст не следует помещать в блоки, блок должен функционально и коротко описывать операцию.

Также можно обозначать ввод/вывод при работе с **файлами** или **сокетами**.

Конструкция цикл



Цикл – разновидность управляющей конструкции в высокоуровневых языках программирования. Предназначена для организации *многократного* исполнения *однотипных инструкций* над *данными*.

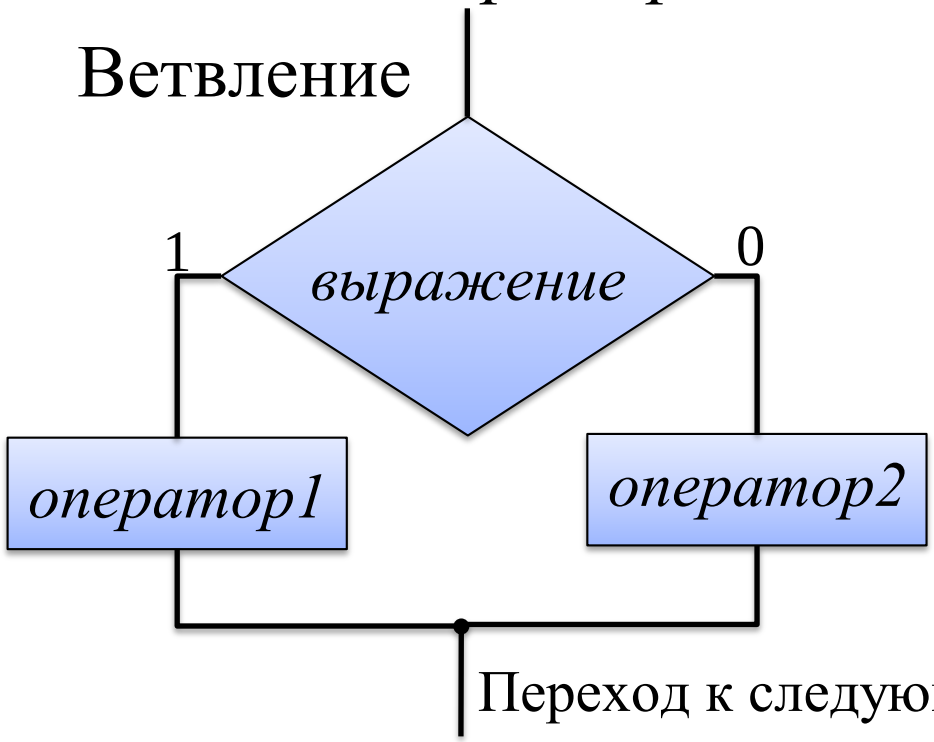
В языке Си предусмотрено 3 циклических конструкции, которые являются *взаимозаменяемыми*:

1) с предусловием – **while**;
2) по счётчику - **for**
3) с постусловием – **do-while;**

Оператор if…else

Общая форма оператора if…else имеет следующий вид:

if (*выражение*)
оператор1
else
оператор2



Если *выражение* **истинно**, то выполняется *оператор1*, а если **ложно** выполняется *оператор2*.

Операторы могут быть как простыми так и составными (блочными)

На блок схемах выражение лучше сокращать, в идеале вписывать в равнобедренный ромб.

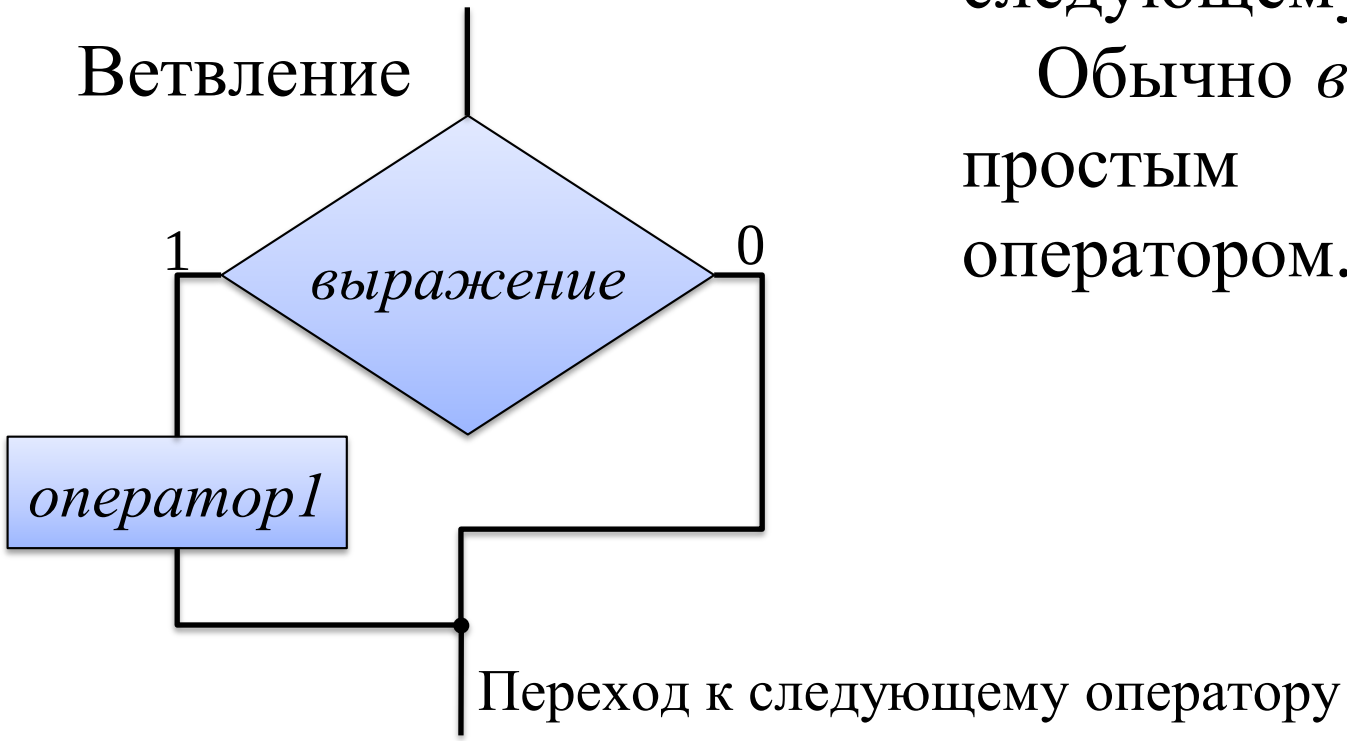
Оператор if else

Оператор if…else может быть записан без else:

if (*выражение*)
оператор1

Если *выражение* **истинно**, то выполняется *оператор1*, а если **ложно** происходит переход к следующему оператору.

Обычно *выражение* является или простым или составным оператором.



Примеры оператора if…else

Пример:

```
if (score > big) printf("Джекпот!\n");
```

Пример с *блочным оператором* :

```
if (joe > ron) {
   joecash++;
   printf("Ты проиграл, Ron.\n");
}
```

Пример с ветвью **else**:

```
if (all_days!= 0)
   printf("%d - общее количество дней: %.lf%% с температурой ниже нуля.\n", all_days, 100.0*(float)cold_days/all_days);
  else
   printf("Данные не введены!\n ");
```

Примеры оператора if…else (продолжение)

Пример **if**…**else** с *блочными операторами*:

```
if (all_days!= 0)
{
   temp = 100.0*(float)cold_days/all_days;
   printf("%d - общее количество дней: %.lf%% с температурой ниже нуля.\n", all_days, temp);
}
else
{
   printf("Данные не введены!\n");
   error++;
}
```

Функции getchar() и putchar()

Функция **getchar**() – возвращает очередной символ входного потока. Не имеет аргументов, а ее результатом является значение типа int.

Функция **putchar**() – отправляет в выходной поток переданный ей целочисленный аргумент типа int.

Пример:

```
char q;
do
{
      q = getchar();
      putchar(q);
} while (q!='0’);
return 0;
```

```
char q;
do
{
      scanf("%c", &q);
      printf("%c",q);
} while (q!='0’);
return 0;
```

Функции специализированы для работы только с символами!

Использование getchar()

```
ch=getchar();  // читать символ
while(ch!='\n’)  // пока не встретится конец строки
{
 ...                    // код обработки символа
 ch=getchar(); // получить следующий символ
}
```

Аналогичная запись

```
while((ch=getchar())!='\n')
```

Аналогичная запись?

```
while(ch=getchar()!='\n')
```

Использование putchar()

```
char ch = '.';
putchar(ch + 1);   // вывести символ
```

```
'.' =>  int(46) => 46+1 = int(47) => '/'
```

46	2E	0010 1110	.
47	2F	0010 1111	/

Для определения различных групп символов в языке Си есть стандартный набор функций для анализа символов, прототипы которых находятся в заголовочном файле ctype.h

Функций проверки символов ctype.h

Имя функции	Возвращает истинное значение, если аргумент является указанным ниже
`isalnum()`	Алфавитно-цифровой (буквенный или цифровой)
`isalpha()`	Алфавитный
`isblank()`	Стандартный пробельный символ (пробел, горизонтальная табуляция либо новая строка) или любой дополнительный символ подобного рода, специфичный для локали
`iscntrl()`	Управляющий символ, такой как <Ctrl+B>
`isdigit()`	Цифра
`isgraph()`	Любой печатаемый символ, отличный от пробела
`islower()`	Символ нижнего регистра
`isprint()`	Печатаемый символ
`ispunct()`	Символ пунктуации (любой печатаемый символ, отличный от пробела или алфавитно-цифрового символа)
`isspace()`	Пробельный символ (символ пробела, новой строки, перевода страницы, возврата каретки, вертикальной или горизонтальной табуляции или, возможно, другой символ, определенный локалью)
`isupper()`	Символ верхнего регистра
`isxdigit()`	Символ шестнадцатеричной цифры

Функций изменения символов ctype.h

Имя функции	Действие
`tolower()`	Если аргумент является символом верхнего регистра, функция возвращает его версию в нижнем регистре; в противном случае она возвращает исходный аргумент
`toupper()`	Если аргумент является символом нижнего регистра, функция возвращает его версию в верхнем регистре; в противном случае она возвращает исходный аргумент

```
char ch = 'h';
ch = toupper(ch); \\ 'H'
ch = tolower(ch); \\ 'h'
```

Множественный выбор else…if

```
if (выражение1)
     оператор1
else if (выражение2)
     оператор2
else if (выражение3)
     оператор3
else
     оператор4
```

Если *выражение1* **истина**, тогда выполняется *оператор1*, иначе происходит переход к проверке *выражения2* и так далее по всем возможным выражениям, иначе выполняется *оператор4*.

Множественный выбор else…if (пример)

```
if ( kwh <= BREAK1)
     bill = RATEl * kwh;
else
     if (kwh <= BREAK2)
          bill = BASE1 + (RATE2*(kwh – BREAK1));
else
     if (kwh <= BREAK3)
          bill = BASE2 + (RATE3*(kwh – BREAK2));
else
     bill = BASE3 + (RATE4 * (kwh – BREAK3));
```

Считается что в операторе **if…else** в части **else** *вложен* оператор другой оператор **if…else**.

Множественный выбор else…if (пример)

```
if ( kwh <= BREAK1)
     bill = RATEl * kwh;
else if (kwh <= BREAK2)
     bill = BASE1 + (RATE2*(kwh – BREAK1));
else if (kwh <= BREAK3)
     bill = BASE2 + (RATE3*(kwh – BREAK2));
else
     bill = BASE3 + (RATE4 * (kwh – BREAK3));
```

Задача - нахождение делителей

```
1  unsigned long num, div;                                  \\#include <stdio.h>
2  bool isPrime;                                            \\#include <stdbool.h>
3   printf("Enter a natural number press q if exit.\n");
4   while (scanf("%lu", &num) == 1){
5       for (div = 2, isPrime = true; (div * div) <= num; div++){
6           if (num % div == 0){
7               if ((div * div) != num)
8                   printf ("%lu is divided by %lu and %lu.\n",num, div, num / div);
9               else
10                  printf("%lu is divided by %lu.\n",num, div);
11              isPrime= false; } }
12    if (isPrime)
13       printf("%lu is a prime number.\n", num);
14    printf("Enter the next number for check; press q if exit.\n");}
15   printf("Buy buy.\n");
16   return 0;}
```

Давайте будем более логичными

Для написания *составных выражений* в операторах **if** и **while и пр.** используются три логических операции:

Операция	Описание	
&&	“И”	**конъюнкция**
||	“ИЛИ”	**дизъюнкция**
!	“НЕ”	**инверсия**

Примеры:

```
exp1 && exp2
exp1 || exp2
!exp1
5>2 && 4>7
5>2 || 4>7
!(4 > 7)
```

```
(практика && время) == (совершенство ||
        экзамен по Программированию)
```

Давайте будем более ~~питоничными~~

C99 вводит альтернативные формы написания логических операций, которые определены в заголовочном файле iso646.h. Включив этот файл в программу, вы можете указывать:

and вместо **&&**

or вместо **||**

not вместо **!**

Традиционное представление	Представление посредством `iso646.h`
`&&`	`and`
`||`	`or`
`!`	`not`

Тернарная функция (операция ?:)

Си предлагает сокращенный способ оператора if…else, для которого по аналогии применяется условная операция ?:.

Это единственная тернарная операция (содержит три операнда)

Пример:

```
x = (y<0) ? -y : y;
```

Что эквивалентно

```
if (y < 0) x = -y;
else x = y;
```

выражение1 **?** *выражение2* **:** *выражение3*

Если *выражение1* имеет истинное (ненулевое) значение, то все условие принимает тоже значение что и *выражение2* **иначе** *выражение3*.

Тернарная функция (примеры)

Запись максимального из двух чисел

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

Запись максимального из трёх чисел

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

Выбор события при сравнении

```
c = a > b ? a - b : a + b;
```

Влияние на выбор при работе с потоком вывода

```
printf("x is %s digit\n", (x % 2 == 0) ? "even" : "odd");
```

Скобки помогут написать верные выражения…

Продолжать и ломать ~~циклы~~

При работе внутри тела цикла, возникает потребность в прерывании или перехода к следующей итерации цикла.
Для этого используются операторы **continue** и **break.**

Оператор **continue** прерывает выполнение итерации и переходит к выполнению следующей итерации перед проверив выражение-условие цикла.

Оператор **break** прерывает выполнение цикла.

Оператор continue (пример)

```
const float MIN = 0.0f;
const float MAX = 100.0f;
float score;
float total = 0.0f;
int n = 0;
float min = MAX;
float max = MIN;
printf("Введите результат первой игры (или q для завершения): ");
while (scanf("%f", &score) == 1)
{
    if (score < MIN || score > MAX)
    {
        printf("%0.1f - недопустимое значение. Повторите попытку: ",
               score);
        continue;  // переход к условию проверки цикла while
    }
    printf("Accepting %0.1f:\n", score);
    min = (score < min)? score: min;
    max = (score > max)? score: max;
    total += score;
    n++;
    printf("Введите результат следующей игры (или q для завершения): ");
}
```

Оператор break (пример)

```
float length, width;

printf("Введите длину прямоугольника:\n");
while (scanf("%f", &length) == 1)
{
    printf("Длина = %0.2f:\n", length);
    printf("Введите ширину прямоугольника:\n");
    if (scanf("%f", &width) != 1)
        break;
    printf("Ширина = %0.2f:\n", width);
    printf("Площадь = %0.2f:\n", length * width);
    printf("Введите длину прямоугольника:\n");
}
```

Оператор множественного выбора switch

```
switch (целочисленное-выражение)
{
    case константа1: операторы1 break;
    case константа2: операторы2 break;
    default: операторы3 break;
}
```

Целочисленное-выражение (включая **char**) оператора **switch** проверяется по *константным значениям* меток **case**. Метки **case** могут быть *выражениями* из констант. Выбирается соответствующий *оператор* и выполняется. Оператор **break** прекращает работу выбранного *оператора* **switch**. Если *выражение* не соответствует ни одной метке **case**, может быть предусмотрена метка **default**, которая выполняется если ни одна из *констант* не соответствует *выражению*.

Оператор множественного выбора (пример)

```
switch (ch)
{
case 'а' :
   printf("архар, дикий горный азиатский баран\n");
   break;
case 'б' :
   printf("бабирусса, дикая малайская свинья\n");
   break;
case 'к' :
   printf("коати, носуха обыкновенная\n");
   break;
case 'в' :
   printf("выхухоль, водоплавающее существо\n");
   break;
case 'е' :
   printf("ехидна, игольчатый муравьед\n");
   break;
case 'р' :
   printf("рыболов, светло-коричневая куница\n");
   break;
   default :
   printf("Вопрос озадачил!\n");
}          /* конец оператора выбора */
```

Дополнительные возможности case-меток

```
switch (ch)
{
    case 'a' :
    case 'A' : a_ct++;
               break;
    case 'e' :
    case 'E' : e_ct++;
               break;
    case 'i' :
    case 'I' : i_ct++;
               break;
    case 'o' :
    case 'O' : o_ct++;
               break;
    case 'u' :
    case 'U' : u_ct++;
               break;
    default  : break;
}
```

На практике перечисляются константные *значения/выражения* через метки **case**, и группы отделяются друг от друга оператором **break**.

Таким образом несколько *констант* соответствуют одному *оператору*

Как видно из практики, оператор **break** применяется не только в циклах, но и в операторе **switch**, в отличии от **continue**.

Дополнительные возможности case-меток

```
switch (ch)
{
    case 'a' :
    case 'A' : a_ct++;
               break;
    case 'e' :
    case 'E' : e_ct++;
               break;
    case 'i' :
    case 'I' : i_ct++;
               break;
    case 'o' :
    case 'O' : o_ct++;
               break;
    case 'u' :
    case 'U' : u_ct++;
               break;
    default  : break;
}
```

На практике перечисляются константные *значения/выражения* через метки **case**, и группы отделяются друг от друга оператором **break**.

Таким образом несколько *констант* соответствуют одному *оператору*

Как видно из практики, оператор **break** применяется не только в циклах, но и в операторе **switch**, в отличии от **continue**.

Операторы switch vs if…else

Основная проблема выбора между конструкций заключается в понимании их + и -.

- **switch** нельзя использовать если выбор строится на *выражении с плавающей запятой* или на *значении переменной*.
- **switch** не используют при больших диапазонах значений

Пример: `if (integer < 1000 && integer > 2)`

- **switch** выполняется быстрее и код выглядит компактнее если задача которую он решает не попадает в первые два пункта.
- Все остальные случаи это оператор **if…else .**

Операторы goto (~~evil~~)

```
goto метка;
…
…
…
 метка: оператор
```

Оператор **goto** приводит к передаче управления оператору, снабженному указанной меткой.
Метки именуются по правилам именования переменных.
Метка может находится где угодно, как до так и после оператора **goto**.

Язык Си не нуждается в операторе такого рода, все остальные управляющие конструкции позволяют разрабатывать программы без использования оператора **goto**.
Следует избегать беспорядочного использования **goto**.

Операторы goto (пример)

```
top: ch = getchar();
     .
     .
     .
if (ch != 'y')
    goto top;
```

```
while (funct > 0)
{
    for (i = 1, i <= 100; i++)
    {
        for (j = 1; j <= 50; j++)
        {
            множество операторов;
            if (признак ошибки)
                goto help;
            операторы;
        }
        еще множество операторов;
    }
    другое множество операторов;
}
третье множество операторов;
help: код устранения ошибки;
```

Задачи на практику

Читать: Главу 7. Управляющие операторы С: ветвление и переходы.
Смотреть вопросы для самоконтроля (288 стр, page 321(pdf))
Упражнения по программированию (290-292 стр, page 325-327(pdf)
Выполнить 3 задания по формуле:

```
char id  = номер_в_списке % 10 +1
const char count_z = 11;
z1 = 1+(count_z%id)
z2 = 4+(count_z %id)
z3 = (7+(count_z %id))% (count_z+1)
```

Задания выполнение не по формуле в зачёт не идут . Но это не мешает вам их выполнять.

Спасибо за внимание

Ревун Артем Леонидович
ст. преп. Кафедры вычислительных систем

Курс «Программирование», осенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)