Язык программирования Си

Бикулов Д.А., Иваницкая Н.В., Иванов А.П.

Семинар 2. Основы синтаксиса языка Си. Типы данных. Определение переменных и констант. Правила видимости. Условный оператор, циклы и операторы управления циклами.

1 Типы данных

Базовыми типами данных в языках Си/Си++ являются следующие:

`char`	символьные данные (одиночный символ), эквивалентен 8-битному целому числу со знаком, которое изменяется в диапазоне от -128 до +127;
`int`	целые числа со знаком, 32 бита (изменяются от -2^{31} до $+2^{31}$);
`float`	вещественные числа одинарной точности, не рекомендуется использовать для сложных расчетов (32 бита под все, приблизительно 7 десятичных знаков после запятой, диапазон изменения экспоненты: ~ 10^{-38} - 10^{+38});
`double`	вещественные числа с двойной точностью (64 бита под все, приблизительно 16 десятичных знаков после запятой, диапазон изменения экспоненты: ~ 10^{-308} - 10^{+308});
`bool`	булевский (логический) тип, переменные могут иметь только два значения: **`true`** (истина) и **`false`** (ложь).

Для целого типа предусмотрен широкий набор вариантов:

`unsigned char`	целые без знака, 8 бит (изменяются от нуля до 255), фактически – «один байт»;
`unsigned int`	целые без знака, 32 бита (изменяются от нуля до 2^{32}-1);
`(unsigned) short int`	целые (без знака), 16 бит (беззнаковые изменяются от нуля до 2^{16}-1);
`(unsigned) long int`	целые (без знака), 64 бит (беззнаковые изменяются от нуля до 2^{64}-1);

Библиотека **`<stdlib.h>`** определяет дополнительный стандартный тип для указания размера в чего-либо в байтах:

`size_t`	целые (без знака), 64 бит.

2 Определение переменных и констант

Имя переменной – комбинация латинских букв, цифр и символа подчеркивания, без пробелов, начинающаяся с буквы:

правильно	неправильно
`a1`	`Боря+Лена`
`B234rty`	`345uyt`

```
            Ars_Longa_Vita_Brevis              John White
```

Каждая переменная должна быть объявлена перед первым обращением к ней, например:

```
int a1; // Каждая инструкция программы  на СИ заканчивается точкой с запятой

char MyName[128];
```

В этот момент переменной выделяется блок оперативной памяти компьютера, достаточный для ее хранения. Эта память будет иметь фиксированный адрес (адрес переменной) – `&a1` в первом случае, `&MyName[0]` во втором случае.

Если переменные объявить приведенным выше способом – то их значение будет совершенно случайным: никакой специальной инициализации значения не происходит!

Так как переменную можно разумно использовать только, когда ее значение конкретно, то переменную лучше инициализировать непосредственно в момент объявления:

```
double temperature = 36.6;
```

Тогда по адресу переменной `&temperature` будет размещено число `36.6`.

На основе базовых типов можно создавать производные, о которых подробно будет рассказано на следующих семинарах:

```
char s[1024];         // строка длиной 1024 символа, массив типа char

char* s1;             // строки неопределенной длины, длина определяется
char s2[];            // по специальному символу в конце строки,
                      // который автоматически добавляется)

double vec[3] = {1.5,2.5,3.5}; // массив с инициализацией значений элементов
```

Переменная может принимать любое значение соответствующего типа, в отличие от константы (постоянной величины). Константы различных типов можно объявлять так:

```
// целая константа:
const int zero = 0, large_num = 7890987;

// вещественные константы:
const double pi = 3.14159;
const double avogadro = 6.02E+23;
const double plank = 6.62e-34;

const char a = 'Q';           // символьная константа
const char s[] = "abcdef";    // строковая константа, спец.символ '\0' в конце
```

3 Арифметические выражения

Арифметические выражения составляются из чисел, переменных, операторов «+», «-», «/», «*», «%», а также других операторов и круглых скобок, при помощи которых можно изменить порядок действий.

Оператор присваивания обозначается символом «=». Его действие заключается в том, что переменная или выражение, находящиеся слева от него, получают значение выражения, которое находится справа от знака:

```
add=S1+rt*(we+qw)-234.5/(ty+oi);
```

К этому моменту значение всех переменных справа от знака «=» должно быть определено, иначе выражение будет вычислено по случайным данным, которые находятся в неинициализированных переменных. Компилятор об этом предупреждает соответствующим предупреждением (warning):

```
the variable S1 is being used without being initialized
```

При написании программ, в частности – в выражениях, применяются следующие виды скобок:

- круглые: `a = b / (3+x)` применяются для изменения порядка действий в выражениях;
- фигурные: `{ ... }` обозначают начало и конец составного оператора также, как применяются операторы `begin` и `end` в Паскале;
- квадратные: `A[15]` или `A[j]` применяются при работе с массивами (переменными с индексами).

4 Правила видимости переменных

Переменной можно пользоваться только после ее объявления. Иногда удобнее объявить все переменные в начале программы или функции (о функциях подробно будет рассказано позже), но это можно сделать в любом месте программы, непосредственно перед использованием переменной.

Необходимо помнить, что в пределах любой функции, в том числе – в пределах функции `main()`, видны и могут использоваться только свои, локальные переменные. Если объявить переменную в пределах блока (внутри фигурных скобок), то она существует и доступна для работы только в пределах этого блока, и перестает существовать в момент выхода из блока.

В частности, если объявить переменную (например, целую переменную `i`) в заголовке цикла `for`:

```
int s = 0;
for (int i = 0; i < 100; i++) { s = s + A[i]; }
```

то необходимо помнить, что она будет доступна для работы только в теле цикла и прекратит свое существование по окончании работы цикла.

5 Условный оператор

Простая форма условного оператора:

```
if ( a < 10 ) a = a + 3; // простое действие, если выражение истинно

if ( a < x ) { // составной блок в качестве действия
    a = a + 3;
    x = x - 8;
}
```

Если логическое выражение в круглых скобках истинно (`true`) – то выполняется указанное действие (простое или составное), если выражение в круглых скобках ложно – то выполняется следующая по порядку инструкция программы.

Альтернативная форма условного оператора:

```
if ( a < 10 ) // простое действие, простая альтернатива
    a = a + 3;
else
    a = a - 5;

if ( d > 0 ) { // составные блоки действия, множественная альтернатива
    x1 = (-b-sqrt(d)) /2 /a;
    x2 = (-b+sqrt(d)) /2 /a;
    cout << "x1 = " << x1 << "x2 = " << x2 << endl << flush;
} else if ( fabs(d) < 1.0E-20 ) {
    x1 = (-b) /2 /a;
```

```
    x2 = x1;
    cout << "x1 = x2 = " << x1 << endl << flush;
} else {
    cout << "net reshenij" << endl << flush;
}
```

Если первое выражение истинно (`true`) – то выполняется первый блок действия, в противном случае проверяются по одному последующие условия (их может быть несколько), если хотя бы одно окажется истинным, выполнится *его* блок действия, если же не окажется истинным ни одно условие – то сработает последний блок, задаваемый `else` без условия.

С помощью условного оператора и оператора перехода на метку `goto` можно организовать любые циклы:

```
int fib1 = 0, fib2 = 1, fib3 = 1, i = 3;
cout << "1: " << fib1 << endl << flush;
cout << "2: " << fib2 << endl << flush;

LABEL1:
fib3 = fib1 + fib2;
cout << i << ": " << fib3 << endl << flush;
fib1 = fib2;
fib2 = fib3;
if ( i < 20 ) goto LABEL1;
```

6 Циклы и операторы управления циклами

Операторы циклов предназначены для удобного и наглядного выполнения повторяющихся действий, возможно, с изменением некоторых параметров в каждой итерации цикла.

6.1 Цикл с параметром for (… ; … ; …)

```
int m = 1, n = 10;

// тело цикла из одного оператора:
for ( int i = 1; i <= n; i = i+1 )
    m = m*i; //  отступаем вправо - так лучше видно вложенность

// то же с составным оператором в качестве тела цикла:
for ( int i = 1; i <= n; i = i+1 )
{
    m = m*i;
}
```

Итерации цикла `for` выполняются с объявленными начальными значениями параметров (первое поле заголовка цикла), каждый раз, пока проверяемое условие (второе поле заголовка) – истинно, модификация для следующей итерации проводится по правилу, указанному в третьем поле заголовка цикла.

Условие проверяется всегда перед самым первым выполнением цикла (то есть, цикл `for` – это цикл с *предусловием*). Если условие цикла ложно, то тело цикла не выполнится вообще ни одного разу.

Особо нужно обратить внимание на то, что как и для условного оператора, точка с запятой после круглой скобки не ставится. Если поставить точку с запятой после круглой скобки:

```
for ( int i = 1; i <= n; i = i+1 ); // ошибка!
    m = m*i;
```

то получится пустой оператор в качестве тела цикла, цикл сработает вхолостую, а затем его тело будет выполнено один-единственный раз, уже для неверных значений его внутренних переменных. То есть возникнет алгоритмическая ошибка, которую не так просто найти, так как компилятор ее не обнаружит.

Нужно обратить внимание на то, что арифметические выражения можно использовать в любом из трех полей внутри круглых скобок заголовка цикла `for`.

Вложенные циклы:

```
int M[n][n];
...

for ( int i = n-2; i >= 0; i = i-1 )
{
    for ( int j = 0; j < n; j = i+1 )
    {
        M[i+1][j] = M[i][j];
    }
}
```

6.2 Цикл с предусловием while (…) { … }

```
int i = 1, m = 1, n = 10;

while (i <= n )
{
    m = m*i;
    i = i+1;
}
```

Цикл `while` выполняется, пока выражение в заголовке цикла (в круглых скобках) истинно. Проверка истинности осуществляется перед выполнением тела цикла, то есть, цикл может не выполниться ни одного раза, если выражение в заголовке цикла сразу ложно.

Распространенной ошибкой любых циклов является зацикливание программы (бесконечный цикл):

```
int i = 1, m = 1, n = 10;

while (i <= n ) // ошибка: переменные не меняются в ходе вычислений,
                // цикл никогда не закончится!
{
    m = m*i;
}
```

6.3 Цикл с постусловием do { … } while (…);

Тело цикла «do {} while» всегда выполняется по меньшей мере один раз – проверка условия производится после *каждого* (в том числе – после первого) выполнения тела цикла:

```
int i = 1, m = 1, n = 10;

do {
    m = m*i;
    i = i+1;
} while ( i < n ); // здесь строгое неравенство. Подумайте, почему?
```

Приводим два фрагмента программы будут работать по-разному. В чем будет выражаться разница?

Пример. Первый фрагмент:

```
#include <stdio.h>

...

char ch;
do {
    ch = getchar();     // считываем символ
    putchar(ch);        // печатаем то, что считали
} while (ch != '\n');
```

Следующий фрагмент будет работать не так, как предыдущий:

```
#include <stdio.h>

...

char ch;
while ( (ch=getchar()) != '\n' ) putchar(ch);
```

В чем причина разницы?

6.4 Операторы управления циклами

Оператор `break` в любой момент прерывает итерации текущего цикла (при вложенных циклах – самого внутреннего) и управление передается оператору, следующему за данным циклом:

```
double m = 1.0;
int n = 100;

for ( int i = 1; i <= n; i = i+1 )
{
    m = m*i;
    cout << i << ": " << m << endl << flush;
    if ( getchar() == '0' )
        break;
}
```

Оператор `continue` немедленно завершает текущую итерацию (но не цикл!) и передает управление на следующую итерацию: для цикла `for` производится изменение параметра цикла, затем – проверка его условия, для циклов `while` – производится только проверка условия:

```
double m = 1.0;
int n = 100;

for ( int i = 1; i <= n; i = i+1 )
{
    m = m*i;
    if ( getchar() == '0' )
        continue;
    cout << i << ": " << m << endl << flush;
}
```

7 *Методы решения уравнений*

Если задано уравнение **`y = f(x) = 0`**, то с помощью компьютера его можно решить приближенно, практически с любой нужной точностью. Для этого можно применить любой из описываемых ниже методов.

Методы касательных и итераций позволяют получить решение гораздо быстрее (за меньшее число итераций), чем методы дихотомии и хорд, однако, применимы они не к любым уравнениям: для того, чтобы их применить, нужно, чтобы выполнялись некоторые условия, накладываемые на производную функции, которая задает левую часть уравнения.

7.1 Метод дихотомии

Метод дихотомии, он же – метод деления отрезка пополам или метод вилки, известен человечеству более 4 тысяч лет. Этот метод заключается в следующем:

1) Получаем от пользователя начальный интервал `[a, b]`, требуемую точность δ и невязку ε.
2) Проверяем, что на концах этого интервала функция правой части уравнения имеет разные знаки: `f(a) * f(b) < 0`?
3) Если нет – то решения на заданном интервале нет, либо оно не единственное. Печатаем диагностику и завершаем программу.
4) Если знаки на концах интервала разные, то находим середину отрезка:

$$c = \frac{a+b}{2}$$

5) Проверяем, на каком из двух интервалов `[a, c]` или `[c, b]` функция меняет знак: `f(a) * f(c) < 0` или `f(c) * f(b) < 0 ?`
6) Если функция меняет знак на первом интервале, то присваиваем переменной `b` значение `c`: `b = c`.
7) Иначе: `a = c`.
8) Повторяем пп. 4-7 до того момента, когда интервал окажется меньше заданной точности δ и одновременно невязка уравнения окажется меньше заданного ε.

Точностью будем называть ширину интервала на последней итерации, который гарантированно накрывает искомый корень.

Невязкой уравнения называется результат подстановки текущего корня `c` в уравнение: `f(c)`.

Таким образом, условие, при котором надо прекратить итерации, записывается так:

```
|b-a| < δ && |f(c)| < ε
```

7.2 Метод хорд

Метод хорд очень похож на метод дихотомии, но работает немного быстрее. Основное отличие заключается в способе разделения отрезка на две части:

1) Получаем от пользователя начальный интервал `[a, b]`, требуемую точность δ и невязку ε.
2) Проверяем, что на концах этого интервала функция правой части уравнения имеет разные знаки: `f(a) * f(b) < 0`?
3) Если нет – то решения на заданном интервале нет, либо оно не единственное. Печатаем диагностику и завершаем программу.
4) Если знаки на концах интервала разные, то находим точку пересечения прямой, соединяющей точки `(a, f(a))` и `(b, f(b))`:

$$c = a + \left| \frac{f(a)}{f(b) - f(a)} \right| (b - a)$$

5) Проверяем, на каком из двух интервалов `[a, c]` или `[c, b]` функция меняет знак: `f(a) * f(c) < 0` или `f(c) * f(b) < 0 ?`
6) Если функция меняет знак на первом интервале, то присваиваем переменной `b` значение `c`: `b = c`.
7) Иначе: `a = c`.
8) Повторяем пп. 4-7 до того момента, когда расстояние между двумя приближениями корня окажется меньше заданной точности δ и одновременно невязка уравнения окажется меньше заданного ε.

Для многих уравнений ширина интервала, на котором ведется поиск корня будет стремиться не к нулю, а к константе, поэтому тут и далее точностью будем называть расстояние между двумя последовательными приближениями корня. Эта величина будет стремиться к нулю при увеличении количества итераций.

7.3 Метод касательных

Метод касательных называется также методом Ньютона, то есть известен уже более 300 лет. Этот метод заключается в следующем:

1) Получаем от пользователя начальное приближение x_0 (номер итерации `i=0`), требуемую точность δ и невязку ε.
2) Любое последующее уточнение значения корня получаем по итеративной формуле:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

   Эта формула получена как точка пересечения касательной в точке x_i к графику функции `f(x)` с осью `OX`.
3) Повторяем п. 2 до того момента, когда ширина интервала $[x_{i+1}, x_i]$ окажется меньше заданной точности δ и одновременно невязка уравнения окажется меньше заданного ε.

Условием сходимости метода касательных является:

$$f(x_0) \cdot f''(x_0) > 0$$

То есть, в области поиска корня уравнения нужно, чтобы функция была постоянно выпуклой или вогнутой.

7.4 Метод итераций

Метод итераций применим и в тех случаях, когда условие сходимости метода касательных не выполняется, но на функцию правой части уравнения может быть наложено другое условие.

Перепишем уравнение в следующем виде:

$$x = F(x)$$

Тогда, если выполняется условие:

$$|F'(x)| < 1$$

То уравнение можно решать так:

1) Получаем от пользователя начальное приближение x_0 (номер итерации $i=0$), требуемую точность δ и невязку ε.
2) Любое последующее уточнение значения корня получаем по итеративной формуле:

$$x_{i+1} = F(x_i)$$

3) Повторяем п. 2 до того момента, когда ширина интервала $[x_{i+1}, x_i]$ окажется меньше заданной точности δ и одновременно невязка уравнения окажется меньше заданного ε.

Условие на производную следует проверять на каждой итерации (т.е. для всех x_i).

Типовое задание: одним из описанных методов решить заданное уравнение.

Программа должна получить начальные значения переменной (начальный интервал для методов дихотомии и хорд, единственное начальное значение для методов касательных и итераций).

После этого программа должна произвести вычисление и напечатать:

1) исходное уравнение, исходный интервал и заданную точность;
2) найденный корень;
3) *точность*: ширину последнего шага итерации (разность между двух последних приближений x);
4) *невязку*: результат подстановки найденного неточного решения в правую часть уравнения;
5) количество проделанных до достижения требуемой точности итераций цикла.

Требуется, чтобы программа была устойчива к задаваемому пользователем интервалу, если корней вообще нет или корень не единственный или не выполняются условия сходимости для методов касательных и итераций – требуется напечатать соответствующую диагностику.

Для методов касательных и итераций производную заданной функции найти аналитически.

От студента требуется написать программу самостоятельно, в настоящее методическое пособие программный код сознательно не включен.

1. Вариант

Решить методом **дихотомии** уравнение:

```
ln(x) - 1/x = 0
```

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности. Функция натурального логарифма в Си: `log(x)`.

2. Вариант

Решить методом **хорд** уравнение:

```
exp(x) + x = 0
```

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности.

3. Вариант

Решить методом **касательных** уравнение:

```
1/x - 2*x = 0
```

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности.

4. Вариант

Решить методом **дихотомии** уравнение:

```
ln(x) + 1 = 0
```

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности. Функция натурального логарифма в библиотеке языка Си: `log(x)`.

5. Вариант

Решить методом **хорд** уравнение:

```
arctg(x) - 1/2 = 0
```

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности. Функция арктангенса в библиотеке языка Си: `atan(x)`.

6. Вариант

Решить методом **касательных** уравнение:

```
exp(x) - 1/2 = 0
```

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности.

7. Вариант

Решить методом **итераций** уравнение:

```
x = 3 - ln(x-3)
```

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности. Функция натурального логарифма в библиотеке языка Си: `log(x)`.

8. Вариант

Решить методом **касательных** уравнение:

```
exp(x) + ln(x) = 0
```

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности. Функция натурального логарифма в библиотеке языка Си: `log(x)`.

9. Вариант

Решить методом **дихотомии** уравнение:

tg(2*x) – 1 - x = 0

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности. Функция тангенса в библиотеке языка Си: `tan(x)`.

10. Вариант

Решить методом **хорд** уравнение:

2*exp(-x) - x = 0

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности.

11. Вариант

Решить методом **касательных** уравнение:

exp(-3*x^2) - x = 0

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности.

12. Вариант

Решить методом **итераций** уравнение:

tg(x+0.1)/3 = x

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности. Функция тангенса в библиотеке языка Си: `tan(x)`.

13. Вариант

Решить методом **дихотомии** уравнение:

exp(-3*x) + 1 - x = 0

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности.

14. Вариант

Решить методом **хорд** уравнение:

exp(-x^2) - x = 0

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности.

15. Вариант

Решить методом **касательных** уравнение:

exp(-x^2) - x^2 = 0

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности.

16. Вариант

Решить методом **касательных** уравнение:

2*exp(-3*x) + 1 - x = 0

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности.

17. Вариант

Решить методом **дихотомии** уравнение:

3*exp(-3*x) - x = 0

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности.

18. Вариант

Решить методом **хорд** уравнение:

$$\exp(-x^3) - 1 - x^3 = 0$$

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности.

19. Вариант

Решить методом **итераций** уравнение:

$$\cos(2*x) = x$$

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности.

20. Вариант

Решить методом **хорд** уравнение:

$$tg(x/2)/10 - 1 - x = 0$$

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности. Функция тангенса в библиотеке языка Си: `tan(x)`.

21. Вариант

Решить методом **касательных** уравнение:

$$3*\cos(x) - x = 0$$

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности.

22. Вариант

Решить методом **итераций** уравнение:

$$\exp(-x) = x$$

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности.

23. Вариант

Решить методом **дихотомии** уравнение:

$$\exp(-3*x) - x = 0$$

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности.

24. Вариант

Решить методом **итераций** уравнение:

$$\exp(-x^2) + 2 = x$$

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности.

25. Вариант

Решить методом **хорд** уравнение:

$$1/x - x/2 = 0$$

Напечатать решение, невязку, ширину интервала на последней итерации и количество итераций, нужных для достижения заданной точности.