Первая программа

Числовые типы данных

Наиболее часто мы будем использовать два простых типа данных **int** и **double**. Тип **int** – для работы с целыми числами, а тип **double** – для работы с вещественными числами.

Стандартные константы `INT_MIN` и `INT_MAX` задают соответственно минимальное и максимальное значения типа **int**. Если в реализации языка программирования Си на хранение значения типа **int** выделяется 4 байта, то эти константы принимают соответственно значения `-2147483648` и `2147483647`. Любое целое число в интервале от `INT_MIN` до `INT_MAX` может быть представлено как значение типа **int**.

Значения типа **double** также называются числами с плавающей точкой двойной точности. Значения данного типа всегда имеют размер 8 байт. Максимальным значением является константа `1.7976931348623157e+308`. Здесь оно приведено с точностью до 16-го знака после запятой. Минимальное значение имеет противоположный знак.

Очевидно, что значений типа **double** конечное число. Поэтому не все вещественные числа в интервале между приведенными константами могут быть представлены в виде значений типа **double**. Более того не все целые числа в этом интервале могут быть представлены в виде значений типа **double**. Однако все целые числа, представимые значениями типа **int**, также могут быть представлены значениями типа **double**.

Работа с типом **double** требует особой аккуратности. В языке Си определены операции сравнения чисел на равенство. Использовать их для типа **double** не всегда является корректным. При попытке подобного сравнения компилятор может выдать предупреждение.

Например, в тексте своей программы мы можем, определить три переменные `x`, `y` и `z` типа **double**. Проинициализировать переменную `x` константой `0.1`, а переменную `y` – константой `0.01`. Затем просуммировать значение переменной `y` десять раз и получившуюся сумму сохранить в переменную `z`.

```
double x = 0.1, y = 0.01, z;
z = y + y + y + y + y + y + y + y + y + y;
```

Если бы мы оперировали вещественными числами, то значения переменных `x` и `z` совпадали бы и равнялись числу `0.1`.

Однако константа `0.1` не может быть точно представлена в виде значения типа **double**. Поэтому при трансляции текста программы в исполняемый файл компилятор заменит данную константу на «ближайшее» к ней значение типа **double**, которое и будет сохранено в переменную `x`.

Арифметические операции над значениями типа **double** выполняются с погрешностью. При этом, по мере выполнения арифметических операций суммарная погрешность растет. В итоге переменные `x` и `z` будут содержать различные значения, и каждое из этих значений будет отличаться от константы `0.1`. Данный пример раскрывает два источника «неточностей», которые нужно учитывать при работе с типом **double** (преобразования, выполняемые компилятором, и арифметические операции).

Поэтому для значений типа **double** вместо выражения вида `x == y` (проверка на точное равенство) рекомендуется использовать выражение вида `fabs(x-y) < 1e-16`. Мы считаем, что два значения типа **double** «совпадают», если модуль их разности не превосходит заданную константу (например, 10^{-16}). Константа может варьироваться в зависимости от решаемой задачи.

Отметим, что для значений типа **int** проверка на точное равенство выполняется без ограничений.

Функции

Наша первая программа продемонстрирует то, как растет вычислительная погрешность

по мере выполнения арифметических операций над значениями типа **double**. Мы будем вычислять сумму `n` первых членов арифметической прогрессии двумя способами – по определению и по известной формуле, и попытаемся найти то значение `n`, при котором эти по разному вычисленные суммы будут отличаться на заданную константу, например, на `1e-7` (10^{-7}).

Ниже приведен фрагмент программы, вычисляющий сумму по определению.

```
1  double sum1(double a, double d, int n)
2  {
3      double sum, cur;
4      int    i;
5
6      sum = 0.;
7      cur = a;
8
9      for(i = 1; i <= n; i ++)
10     {
11         sum += cur;
12         cur += d;
13     }
14
15     return sum;
16 }
```

Данный фрагмент содержит определение функции `sum1`. Определение функции всегда включает в себя заголовок (строка 1) и тело функции (строки 2-16).

Структура заголовка следующая. Сначала указывается тип возвращаемого значения. Если функция не возвращает никаких значений, то указывается ключевое слово **void**. Далее, идет идентификатор (имя) функции и ограниченный круглыми скобками список параметров. Если функция не имеет параметров, то в круглых скобках ставится ключевое слово **void**. Если параметров больше одного, то они отделяются друг от друга запятыми. Для каждого параметра указывается его тип и идентификатор.

Как видно функция `sum1` возвращает значение типа **double** и содержит три параметра. Параметр `a` имеет тип **double** и интерпретируется как начальный член арифметической прогрессии. Параметр `d` имеет тип **double** и интерпретируется как разность арифметической прогрессии. Наконец, параметр `n` имеет тип **int** и задает сколько первых членов арифметической прогрессии функция `sum1` должна просуммировать.

Тело функции всегда ограничивается фигурными скобками `{` и `}`. В начале определяются локальные переменные (если они требуются). Строка 3 содержит определение переменных `sum` и `cur` типа **double**. Строка 4 содержит определение переменной `i` типа **int**.

Переменная `sum` рассматривается как сумма `i` первых членов арифметической прогрессии. В строке 6 данная переменная инициализируется нулем. Мы это интерпретируем как сумму нуля первых членов арифметической прогрессии. Переменная `cur` рассматривается как текущий член арифметической прогрессии. В строке 7 данная переменная инициализируется первым членом прогрессии.

В строках 9–13 фигурирует оператор цикла **for**. Данный оператор имеет следующую структуру.

```
for (<init>; <test>; <incr>) <body>
```

Оператор `<body>` называется телом цикла. Смысл цикла состоит в многократном выполнении этого оператора. Подобное выполнение называется итерацией цикла. В начале выполнения цикла однократно вычисляется инициализирующее выражение `<init>`. Далее, перед очередной итерацией вычисляется проверочное выражение `<test>`. Если вычисленное значение равно нулю (нулевое значение интепретируется как «ложь», а ненулевое – как «истина»), то цикл завершается. В противном случае выполняется очередная итерация цикла. После каждой итерации цикла сразу же вычисляется выражение перехода `<incr>`.

В рассматриваемом примере цикла инициализирующим выражением является `i = 1`. Данное выражение присваивает переменной `i` значение `1`. Проверочное выражение `i <= n` будет истинным, если значение переменной `i` меньше либо равно значению переменной `n`. Выражение перехода `i ++` увеличивает текущее значение переменной `i` на `1`.

Телом данного цикла является составной оператор строки 10-13. В строках 11 и 12 фигурирует сокращенная операция `+=` (выражение вида `a = a + b` всегда может быть заменено на выражение `a += b`). В строке 11 значение переменной `sum` увеличивается на текущее значение переменной `cur`. Новым значением переменной `sum` становится сумма `i` первых членов арифметической прогрессии. Аналогично, в строке 12 значение переменной `cur` увеличивается на величину `d`. Новым значением переменной `cur` становится `i + 1` член арифметической прогрессии.

После завершения выполнения цикла переменная `sum` будет содержать сумму первых `n` членов арифметической прогрессии, а переменная `cur` будет содержать `n+1` член прогрессии.

Строка 15 содержит оператор **return**. Оператор **return** вызывает завершение выполнения функции и возврат значения, указанного в операторе. В данном случае будет возвращено значение переменной `sum`.

Ниже представлена альтернативная реализация функции `sum1`. В данной реализации инициализация переменных `sum` и `cur` перенесена в инициализирующее выражение цикла **for**. Выражение `cur += d` перенесено из тела цикла в выражение перехода.

```
1  double sum1(double a, double d, int n)
2  {
3      double sum, cur;
4      int    i;
5
6      for(i = 1, sum = 0., cur = a; i <= n; i ++, cur += d)
7          sum += cur;
8
9      return sum;
10 }
```

Ниже приведена реализация функции `sum2`, вычисляющая сумму `n` первых членов арифметической прогрессии по формуле. Тело данной функции состоит из одного оператора **return**.

```
1  double sum2(double a, double d, int n)
2  {
3      return (2 * a + (n - 1) * d) * n / 2;
4  }
```

Ввод и вывод

Как мы смогли увидеть язык программирования Си позволяет пользователям определять собственные функции. Помимо этого, стандартная библиотека языка Си содержится целый набор стандартных функций. В том числе, функцию `printf` – для вывода данных на экран (печати в стандартный поток вывода), и функцию `scanf` – для считывания данных с клавиатуры (чтения из стандартного потока ввода).

Простейший пример использования функции `printf`.

```
printf("My first program.\n");
```

Данный вызов выведет на экран следующую строку.

```
My first program.
```

Рассмотрим более сложный пример вызова функции `printf`.

```
int x = 1;
double y = 1.5;
printf("x = %d x = %5d y = %f y = %.1f y = %5.2f\n", x, x, y, y, y);
```

Данный вызов выведет на экран следующую строку.

```
x = 1 x =       1 y = 1.500000 y = 1.5 y =   1.50
```

Первый параметр – это строка формата печати (последовательность символов, заключенных в двойные кавычки). Данная строка может содержать обычные символы, которые будут просто печататься, а также спецификации формата печати и специальные последовательности символов. Например, последовательность `\n` является специальной последовательностью и задает переход на новую строку.

Спецификации формата печати начинаются с символа `%`. Если спецификация заканчивается символом `d`, то она используется для печати значений типа **int**. Если спецификация заканчивается на `f`, то она используется для печати значений типа **double**. В вызове функции `printf` второй и последующий параметры задают значения, которые должны быть напечатаны при помощи соответствующей спецификации (первая спецификация соответствует второму параметру, вторая спецификация – третьему параметру и т.д.).

После символа `%` может стоять число, которое задает минимальную ширину поля печати. Далее, может следовать `.` и число, задающее точность. Для значений типа **double** это количество цифр после запятой, которые будут напечатаны. По умолчанию точность равна 6.

Ниже приведен пример использования функции `scanf` для считывания двух значений типа **double** и сохранения их в переменных `a` и `d`.

```
double a, d;
scanf("%lf%lf", &a, &d);
```

Первый аргумент представляет собой строку, содержащую спецификаторы ввода. Далее, следуют адреса переменных, в которые должны быть записаны считаные значения. Для получения адреса переменной используется унарная операция `&`.

Функция `scanf` возвращает число считанных значений. В рассматриваемом примере могут быть возвращены числа 2, 1, 0 или константа `EOF` (обычно это `-1`). Результат, возвращаемый функцией `scanf`, всегда нужно отслеживать. Если не все значения были корректно считаны, то необходимо соответствующим образом реагировать. Поэтому рассмотренный пример было бы правильно модифицировать следующим образом.

```
double a, d;
if(scanf("%lf%lf", &a, &d) != 2)
{
    // Treat error.
}
```

Здесь используется условный оператор **if**, который имеет следующую структуру.

```
if(<test>) <op>
```

Выполнение условного оператора заключается в вычислении проверочного выражения `<test>`. Если вычисленное значение отлично от нуля («истина»), то выполняется оператор `<op>`. Иначе (ноль трактуется как «ложь») никаких действий не выполняется.

Оператор **if** является частным случаем условного оператора **if-else**, который имеет следующую структуру.

```
if(<test>) <op1> else <op2>
```

Выполнение условного оператора заключается в вычислении проверочного выражения `<test>`. Если вычисленное значение отлично от нуля, то выполняется оператор `<op1>`. Иначе выполняется оператор `<op2>`.

Функция `main`

В программе на языке Си должна присутствовать функция с именем `main`. С данной функции начинается выполнение программы. Функция `main` должна возвращать значение

типа **int**. Это значение отслеживается операционной системой. Ноль говорит об успешном выполнении программы, а ненулевое значение сигнализирует об ошибке.

Рассмотрим пример функции `main`.

```
1  int main(void)
2  {
3      double a, d, s1, s2, r;
4      int    n;
5
6      printf("Input a: ");
7      if(scanf("%lf", &a) != 1)
8      {
9          printf("Can't read a ...\n");
10         return -1;
11     }
12
13     printf("Input d: ");
14     if(scanf("%lf", &d) != 1)
15     {
16         printf("Can't read d ...\n");
17         return -1;
18     }
19
20     for(n = 1; n <= MAX; n ++)
21     {
22         s1 = sum1(a, d, n);
23         s2 = sum2(a, d, n);
24         r  = s1 - s2;
25         r  = r > 0 ? r : -r;
26
27         if(r > EPS)
28         {
29             printf("Find n: %d\n", n);
30             break;
31         }
32         else
33             printf("n: %5d s1: %20.9f s2: %20.9f r: %.12f\n", n, s1, s2, r);
34     }
35
36     return 0;
37 }
```

В данном примере функция `main` не имеет параметров. Строка 3 содержит определения локальных переменных типа **double**. В переменную `a` будет сохранен первый член арифметической прогрессии, а в переменную `d` – ее разность. Переменная `s1` будет использоваться для хранения суммы `n` первых членов арифметической прогрессии, вычисленной по определению. Переменная `s2` будет использоваться для хранения суммы `n` первых членов арифметической прогрессии, вычисленной по формуле. В переменную `r` будет сохраняться модуль разности величин `s1` и `s2`.

Строка 4 содержит определение переменной `n` типа **int**.

Строка 6 содержит вызов функции `printf`, которая напечатает в стандартный поток вывода приглашение ввести первый член прогрессии.

В строках 7-11 осуществляется считывание значения типа **double** из стандартного потока ввода и его запись в переменную `a`. Если во время считывания произошла ошибка, то вызывается функция `printf` (строка 9) для печати в стандартный поток вывода сообщения об ошибке. После этого вызывается оператор **return** (строка 10) для досрочного завершения программы с кодом ошибки `-1`.

Строка 13 содержит вызов функции `printf`, которая напечатает в стандартный поток вывода приглашение ввести разность арифметической прогрессии.

В строках 14-18 осуществляется считывание значения типа **double** из стандартного потока ввода и его запись в переменную `d`. Если во время считывания произошла ошибка, то вызывается функция `printf` (строка 16) для печати в стандартный поток вывода сообщения об ошибке. После этого вызывается оператор **return** (строка 17) для досрочного завершения программы с кодом ошибки `-1`.

Внутри функции `main` используются две константы `MAX` и `EPS`, которые будут определены вне этой функции. Константа `MAX` – это количество попыток, которое мы отводим для нахождения искомого значения `n`. Искать это значения мы будем последовательным перебором, начиная с `n = 1` и до `MAX`. С этой целью будет использоваться цикл **for** (строки 20-34).

В теле цикла вызывается функция `sum1` (строка 22) для нахождения суммы `n` первых членов прогрессии по определению. Найденное значение записывается в переменную `s1`. Далее, вызывается функция `sum2` (строка 23) для нахождения суммы `n` первых членов прогрессии по определению. Найденное значение записывается в переменную `s2`.

В строках 24-25 вычисляется модуль разности величин `s1` и `s2`, который сохраняется в переменную `r`.

В строке 25 используется тернарная условная операция `?:`, которая имеет следующую структуру.

```
<val1> ? <val2> : <val3>
```

Если значение `<val1>` отлично от нуля («истина»), то результатом операции будет значение `<val2>`. В противном случае результатом операции будет значение `<val3>`.

Далее, следует условный оператор **if-else** (строки 27-33). Если значение переменной `r` больше константы `EPS` (строка 27), то мы нашли искомое значение `n`. В этом случае мы вызываем функцию `printf` (строка 29) для печати найденного значения. После этого переходим к оператору **break** (строка 30) для принудительного выхода из цикла.

Если значение переменной `r` меньше константы `EPS`, то мы вызываем функцию `printf` (строка 33) для печати текущих значений переменных `n`, `s1`, `s2`, `r` и переходим к очередной итерации цикла, увеличивая значение переменной `n` на `1`.

После выполнения цикла вызывается оператор **return** со значением `0`, говорящем о том, что программа была выполнена успешно.

Общая структура программы

Наша первая программа будет иметь следующую структуру.

```
1  #include <stdio.h>
2
3  #define EPS 1e-7
4  #define MAX 100000
5
6  double sum1(double a, double d, int n);
7  double sum2(double a, double d, int n);
8
9  int main(void)
10 {
11     ...
12 }
13
14 double sum1(double a, double d, int n)
15 {
16     ...
17 }
18
19 double sum2(double a, double d, int n)
20 {
21     ...
22 }
```

Вместо строки 11 необходимо вставить содержимое функции `main`, вместо строки 16 – содержимое функции `sum1`, а вместо строки 21 – содержимое функции `sum2`.

Внутри функции `main` были использованы (вызывались) функции `sum1`, `sum2`, `printf` и `scanf`. До своего первого использования функция должна быть описана. Подобное описание называется прототипом и включает в себя заголовок функции и символ `;`. В строке 6 описан прототип функции `sum1`, а в строке 7 – функции `sum2`. Стандартные функции `printf` и `scanf` описаны в заголовочном файле `stdio.h`.

В строках 1, 3, 4 – записаны директивы препроцессора. Перед основной фазой компиляции в текст программы будет добавлено содержимое файла `stdio.h`. Любое появление в тексте программы идентификатора `EPS` (`MAX`) будет заменено на последовательность символов `1e-7` (`100000`).

Компиляция и выполнение

Предположим, что текст нашей первой программы был сохранен в файле `first.c`. Для компиляции необходимо выполнить команду.

```
gcc first.c -o first
```

В случае успеха компиляции будет создан исполняемый файл `first`. Запустим его

```
./first
```

Рекомендуется ввести следующие значения `a = 1.5` и `d = 0.001`.

В компьютерном классе компилятор автоматически использует ряд дополнительных опций. Компилятор выдает больше предупреждений, которые к тому же трактуются как ошибки. В «домашних» условиях для компиляции рекомендуется использовать следующую команду.

```
gcc -Wall -Wextra -Werror -Wold-style-declaration -Wold-style-definition \
    -Wfloat-equal -pedantic -std=c99 first.c -o first
```