Урок 5. Циклы while и do…while

На предыдущем уроке Вы познакомились с циклом `for`. Хотя этот цикл универсален, но на практике часто удобнее и нагляднее использовать два других цикла: `while` и `do…while`. Сначала рассмотрим оба цикла в общем виде, а затем покажем их работу на примерах.

Цикл `while` (с предусловием)

```
while (условие)
{
  тело_цикла
}
```

Этот оператор называется **оператором цикла с предусловием**, т.е. **условие проверяется перед выполнением тела цикла**. Цикл выполняется, пока условие истинно. Если при первой проверке условия оно оказывается ложным, то тело цикла выполнено не будет.

Ключевое слово while переводится как «пока»

Алгоритм работы:

1. Если условие истинно, перейти к п.2, иначе – выйти из цикла
2. Выполнить тело цикла
3. Перейти к п. 1

Цикл `do…while` (с постусловием)

```
do {
  тело_цикла
} while (условие);
```

Этот оператор называется **оператором цикла с постусловием**, т.е. **условие проверяется в конце цикла**, а следовательно тело цикла выполняется хотя бы один раз. Цикл выполняется, пока условие истинно.

Ключевые слова do…while переводятся как «делать…пока».

Алгоритм работы:

1. Выполнить тело цикла
2. Если условие истинно, перейти к п.1.
3. Выйти из цикла

Задание 1

Написать программу, которая запрашивает пароль (целое число) до тех пор, пока не будет введён правильный пароль

Решение

```
#include <stdio.h>

void main()
{
  int pwd, inp;
  pwd = 1024;

  do {                    // Начало цикла
    printf("Enter password:");
    scanf("%d", &inp);
  } while (pwd != inp); // Конец цикла

  printf("Welcome!\n");
  while(!kbhit());
}
```

Пояснения к тексту программы

`int pwd, inp;`

Объявление двух целочисленных переменных. В переменной `pwd` хранится правильный пароль, а в `inp` – пароль, введённый пользователем.

`pwd = 1024;`

Правильный пароль помещается в переменную `pwd`

```
do {                    // Начало цикла
  printf("Enter password:");
  scanf("%d", &inp);
} while (pwd != inp); // Конец цикла
```

В теле цикла происходит ввод пароля с клавиатуры. В условии цикла происходит сравнение введённого пароля с правильным. Если они не совпадают, то тело цикла снова выполняется. Если введён правильный пароль, то цикл завершается.

`while(!kbhit());`

В этом цикле происходит ожидание нажатия клавиши. Если нажата клавиша, то функция `kbhit()` возвращает ненулевое значение. Условие `!kbhit()` можно записать как `kbhit()==0`. Точка с запятой в конце означает, что тело цикла пусто.

Задание 2.1

Написать программу для вычисления суммы чисел от 1 до 100, используя цикл `while`.

Решение

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

void main()
{   int sum = 0, i = 1;

    while(i <= 100) // Начало цикла
    {
        sum += i;
        i++;
    }               // Конец цикла

    printf("Sum: %d\n", sum);
    while(!kbhit());
}
```

Пояснения к тексту программы

`#include <conio.h>`

Подключение библиотеки `conio.h`. Её название conio расшифровывается как console input-output – консольный ввод-вывод. Она содержит дополнительные функции ввода-вывода для работы в текстовом режиме (консоли). В этой программе используется функция `kbhit` из этой библиотеки (проверяет, нажата ли клавиша). Ранее мы не подключали её явно и компилятор задействовал её автоматически при использовании `kbhit`.

```
int sum = 0, i = 1;
```

В этой строке происходит объявление и инициализация переменных. Это можно записать и другим способом:

```
int sum, i;
sum = 0;
i = 1;
```

```
while(!kbhit());
```

Напоминаем, что в этом цикле происходит ожидание нажатия клавиши. Для наглядности этот цикл можно записать развёрнуто:

```
while (kbhit() == 0) {}
```

Задание 2.2

Написать программу для вычисление суммы чисел от 1 до 100, используя цикл `do...while`

Решение

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

void main()
{   int sum = 0, i = 1;

    do
    {
        sum += i;
        i++;
    } while (i <= 100);

    printf("Sum: %d\n", sum);
    while(!kbhit());
}
```

Основное отличие от программы в предыдущем задании - использование цикла с постусловием (`do...while`), а не с предусловием (`while`).

Задание 3

Дана последовательность – первое и второе числа = 1, третье и все остальные числа последовательности получаются сложением двух предыдущих (числа Фибоначчи). Последовательность выглядит следующим образом:

1,	**1,**	**2,**	**3,**	**5,**	**8,**
		(1+1)	(1+2)	(2+3)	(3+5)

Напечатать все числа последовательности, меньшие A.

Решение

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

void main()
{ // Объявление и инициализация переменных
  int a;             // Верхняя граница a
  int p = 1, q = 1; // Первые 2 числа Фибоначчи
  int t;             // Временная переменная
  // Вывод подсказки
  printf("Программа выводит числа Фибоначчи,\n");
  printf("которые меньше числа a\n");
  // Запрос числа a
  printf("Введите a: ");
  scanf("%d", &a);
  // Вывод первого числа Фибоначчи
  printf("%d\n", p);
  // Цикл
  do {
    // Вывод числа Фибоначчи
    printf("%d\n", q);
    // Расчёт следующего числа
    t = p + q;
    p = q;
    q = t;
  } while (q < a);
  // Ожидание нажатия клавиши
  while(!kbhit());
}
```

Пояснения к тексту программы

```
int p = 1, q = 1; // Первые 2 числа Фибоначчи
```

В этой строке объявляются переменные для хранения двух последних чисел Фибоначчи. В них помещаются первые два числа Фибоначчи, которые будут служить отправной точкой для дальнейших расчётов.

```
// Запрос числа a
printf("Введите a: ");
scanf("%d", &a);
```

Здесь происходит запрос числа, которое служит верхней границей для выводимых чисел Фибоначчи

```
// Вывод первого числа Фибоначчи
printf("%d\n", p);
```

На экран выводится первое число Фибоначчи. Остальные будут выведены уже в цикле do…while. Далее разберём цикл оператор за оператором.

```
do {
```

Начало цикла

```
{ // Вывод числа Фибоначчи
  printf("%d\n", q);
```

Печать вновь найденного числа Фибоначчи. При первом выполнении тела цикла будет выведено второе число (единица).

```
  // Расчёт следующего числа
  t = p + q;
  p = q;
  q = t;
```

Сначала вычисляется новое число Фибоначчи и помещается в переменную `t`. Далее последнее число становится предпоследним и перемещается из переменной `q` в переменную `p`. Затем найденное число Фибоначчи из переменной `t` помещается в переменную `q` (т.е. оно становится последним). Всё готово к вычислению ещё одного числа из этой последовательности.

```
} while (q < a);
```

Если найденное число Фибоначчи меньше верхнего предела, то продолжить выполнение цикла, иначе – выйти из цикла.

Задание 4*

Написать игру «Угадай число»: программа «задумывает» число от 1 до 1000, а пользователь угадывает. После каждой попытки программа может выдавать три сообщения:

- «Перелёт» (введённое число больше задуманного)
- «Недолёт» (введённое число меньше задуманного)
- «Вы угадали!»

Указание: в библиотеке `stdlib.h` есть функция `rand`, которая возвращает случайное число от `0` до `RAND_MAX`. Для сброса генератора случайных чисел используйте функцию `srand`. На вход можно подать выход функции `time` из библиотеки `time.h` (см. урок 4, задание 7*)

Решение

```
// Подключение библиотек
#include <stdio.h>  // Стандартный ввод-вывод
#include <stdlib.h> // Функции общего назначение
#include <time.h>   // Работа со временем
#include <conio.h>  // Расширенный ввод-вывод

void main()
{ int num, inp, t;
  // Вывод информации о игре
  printf("Игра 'УГАДАЙ ЧИСЛО'\n");
  printf("Компьютер загадывает число от 1 до 1000,\n");
  printf("а Вы угадываете\n\n");
  // "Задумывание" числа
  t = time(NULL); // Получить текущее время
  srand(t);       // Сбросить генератор
                  // случайных чисел
  num = 1 + rand() * 999 / RAND_MAX; // Случайное число
                                     // от 1 до 1000
  // Игровой цикл
  do {
    // Ввод числа с клавиатуры
    printf("Введите число: ");
    scanf("%d", &inp);
    // Вывод подсказки
    if (inp < num)
    {
      printf("Недолёт!\n");
    }
    else if (inp > num)
    {
      printf("Перелёт!\n");
    }
  } while (inp != num);
  // Число угадано
  printf("Вы угадали!\n"); // Сообщение
  while(!kbhit());         // Ожидание ввода с клавиатуры
}
```

Пояснения к тексту программы

```
// "Задумывание" числа
t = time(NULL); // Получить текущее время
srand(t);       // Сбросить генератор
                // случайных чисел
```

Сбросить генератор случайных чисел с помощью функции `srand` из `stdlib.h`. Для сброса используется текущее время, полученное от функции time (число секунд с полуночи 1 января 1970 года). Если его не сбросить, то игра будет «задумывать» каждый раз одно и то же число.

```
num = 1 + rand()*999 / RAND_MAX; // Случайное число
                                 // от 1 до 1000
```

Получение случайного числа от 1 до 1000. Функция `rand()` возвращает случайное число в диапазоне от `0` до `RAND_MAX`, поэтому необходимы арифметические преобразования.

```
do {
```

Начало игрового цикла

```
if (inp < num)
{
  printf("Недолёт!\n");
}
else if (inp > num)
{
  printf("Перелёт!\n");
}
```

В этом условном операторе происходит вывод подсказки игроку – если введено число меньше задуманного, то печатается «Недолёт», если больше – то «Перелёт»

```
} while (inp != num);
```

Если игрок не угадал число (введённое число не равно «задуманному»), то дать ещё одну попытку.

Задание 5*

Написать программу для вычисления числа π с заданной точностью, используя следующую формулу:

$$\sqrt{3}\frac{\pi}{6} = \sum_{i=0}^{\infty} \frac{(-1)^i}{(2i+1)2^i} = 1 - \frac{1}{3}\cdot\frac{1}{3} + \frac{1}{5}\frac{1}{3^2} - \frac{1}{7}\frac{1}{3^3} + \ldots$$

Используйте цикл `do...while`

Для проверки: π≈3,141 592 653 589 793

Решение

```
#include <stdio.h> // Стандартный ввод-вывод
#include <math.h>  // Математические функции
#include <conio.h> // Расширенный ввод-вывод

void main()
{
  double pi = 0, x;
  int i = 0;

  do {
    // Рассчитать следующий член ряда
    x = 1.0 / (2.0*i + 1.0) / pow(3.0, i);
    // Добавить член ряда к сумме (число Пи)
    if (i % 2 == 0)
      pi += x;
    else
      pi -= x;

    i++;
    // Вывести новое приближение
    printf("%.10lf\n", pi*6.0/sqrt(3.0));
  } while (x >= 1E-8);
  // Ожидать нажатия клавиши
  while(!kbhit());
}
```

Пояснения к тексту программы

```
double pi = 0, x;
int i = 0;
```

Объявление и инициализация переменных. В переменной pi будет храниться приближенное значение π, а в переменной i – номер члена ряда (см. формулу).

```
do {
```

Начало цикла

```
// Рассчитать следующий член ряда
x = 1.0 / (2.0*i + 1.0) / pow(3.0, i);
```

Рассчитывает член ряда с номером i по формуле $\frac{(-1)^i}{(2i+1)2^i}$ (при этом не учитывается $(-1)^i$)

Функция `pow` из библиотеки `math.h` – возведение в степень (`pow(3.0, i)` – это 3^i). **Обратите внимание, что числа записаны как 1.0, 2.0, 3.0 – они типа `double`, а не `int` (если писать 1, 2, 3, то правильного ответа не получится).**

```
// Добавить член ряда к сумме (число Пи)
if (i % 2 == 0)
  pi += x;
else
  pi -= x;
```

Здесь происходит добавление члена ряда к сумме с учётом $(-1)^i$: если условие `i%2==0` истинно (`i` – чётно, т.е. остаток от деления на 2 равен нулю), то новый член прибавляется к `pi`, если нет – то отнимается от `pi`.

```
i++;
```

Переход к следующему члену ряда

```
printf("%.10lf\n", pi*6.0/sqrt(3.0));
```

Вывести вновь полученное приближенное значение числа π на экран (выводится 10 знаков после запятой, из которых только 7-8 надёжных). Домножение на $\frac{6}{\sqrt{3}}$ требуется по формуле.

```
} while (x >= 1E-8);
```

Если вновь прибавленный член ряда меньше заданной погрешности, то расчёт останавливается, если же нет – то выполнить тело цикла ещё раз (новая итерация)

Задание 6*

Выполните задание 5, но без использования ключевого слова `if` и оператора ==. Использовать pow(-1,i) тоже не разрешается.

Решение

```
#include <stdio.h> // Стандартный ввод-вывод
#include <math.h>  // Математические функции
#include <conio.h> // Расширенный ввод-вывод

void main()
{
  double pi = 0, x;
  int i = 0;

  do {
    // Рассчитать следующий член ряда
    x = 1.0 / (2.0*i + 1.0) / pow(3.0, i);
    // Добавить член ряда к сумме (число Пи)
    pi += i++ % 2 ? -x : x;
    // Вывести новое приближение
    printf("%.10lf\n", pi*6.0/sqrt(3.0));
  } while (x >= 1E-8);
  // Ожидать нажатия клавиши
  while(!kbhit());
}
```

Пояснения к тексту программы

Почти все строки этой программы были детально разобраны в задании 5, но один новый оператор заслуживает отдельного рассмотрения:

```
pi += i++ % 2 ? -x : x;
```

Здесь использовано **условное выражение**, который в общем виде записывается так:

```
условие ? выражение_1 : выражение_2
```

Если *условие* перед знаком вопроса истинно, то значение всего выражения – *выражение_1*, иначе – *выражение_2*.

В нашем случае это означает следующее: если выражение `i++%2` истинно (не равно нулю) и значение переменной `i` чётное, то вернуть `-x`, иначе – вернуть `x`.

`i++` - **оператор инкремента в постфиксной форме** (значение переменной `i` будет увеличено на единицу после выполнения всей строки – подробнее см. урок 4, задание 8**)

Можно использовать и другие варианты, например:

```
pi += ++i % 2 ? x : -x;
pi += !(i++ % 2) ? x : -x;
pi += ++i & 1 ? x : -x;
pi += i++ & 1 ? -x : x;
```

Внимание! Умеренно используйте условное выражение (например, оно неуместно внутри сложного арифметического выражения) и не злоупотребляйте префиксными/постфиксными формами операторов инкремента/декремента: это затрудняет чтение и понимание текста программы! Не нужно заставляйте гадать читателя, что такое `x+++y`.

Задание 7*

Напишите программу для нахождения квадратного корня числа *x*, не используя функцию `sqrt` из `math.h`. Используйте такой алгоритм:

1. В качестве начального приближения *sqr* взять 1
2. Сохранить старое приближение (*sqr_old=sqr*)
3. Найти новое приближение $sqr = \frac{x/sqr + sqr}{2}$
4. Если $|sqr\_old - sqr\_new| \geq accuracy$ (*accuracy* – абсолютная погрешность или точность), то перейти к пункту 2 (новая итерация)
5. Завершить работу

Решение

```
/* Вычисление квадратного корня */
#include <stdio.h>
#include <conio.h>
#include <math.h>

void main()
{
  double x, sqr, sqr_old;
  // Ввод исходного числа
  printf("x: ");
  scanf("%lf", &x);
  /* Начальное приближение
     для квадратного корня */
  sqr = 1;

  /* Цикл, вычисляющий корень */
  do {              // Начало цикла
    sqr_old = sqr; // Сохранить старое приближение
    sqr = (x/sqr + sqr) / 2.0; // Получить новое приближение
    printf("%.8lf\n", sqr);    // Вывести новое приближение
  } while (fabs(sqr_old-sqr)>=1E-9); // Условие точности

  // Вывод окончательного приближения
  printf("sqrt(x) = %.8lf\n", sqr);
  // Ожидание нажатия клавиши
  while (!kbhit());
}
```

Такой метод расчёта называется итерационным (метод последовательных приближений). Вычисление останавливается при достижении нужной точности (разность между старым и новым значением меньше заданной. Функция `fabs` из библиотеки `math.h` вычисляет модуль числа типа `double`.

Резюме

Цикл `while` (с предусловием)

```
while (условие)
{
  тело_цикла
}
```

Этот оператор называется **оператором цикла с предусловием**, т.е. **условие проверяется перед выполнением тела цикла**. Цикл выполняется, пока условие истинно. Если при первой проверке условия оно оказывается ложным, то тело цикла выполнено не будет.

Пример:

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

void main()
{   int sum = 0, i = 1;
    while(i <= 100)
    {
        sum += i;
        i++;
    }
    printf("Sum: %d\n", sum);
    while(!kbhit());
}
```

Цикл `do`…`while` (с постусловием)

```
do {
  тело_цикла
} while (условие);
```

Этот оператор называется **оператором цикла с постусловием**, т.е. **условие проверяется в конце цикла**, а следовательно тело цикла выполняется хотя бы один раз. Цикл выполняется, пока условие истинно.

Пример:

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

void main()
{
  int pwd, inp;
  pwd = 1024;
  do {
    printf("Enter password:");
    scanf("%d", &inp);
  } while (pwd != inp);
  printf("Welcome!\n");
  while(!kbhit());
}
```

Упражнения

1. Вычислите факториал числа ($N! = 1 \cdot 2 \cdot 3 \cdot \ldots (N-1) \cdot N$), используя цикл `while`
2. Вычислите факториал числа, используя цикл `do…while`
3. Найти сумму ряда для заданного N: $1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{N}$, используя `while` или `do…while`
4. Дана последовательность сумм: 1; 1+2; 1+2+3; найти первый член последовательности, превосходящий заданное число D
5. Найти первую степень 2, превышающую заданное число A
6. Добавьте счётчик попыток в игру «Угадай число» (см. задание 4*).

Упражнения повышенной трудности

1. Напишите программу для извлечения кубического корня.
   Указание: достаточно лишь слегка изменить алгоритм задания 7 – замените `sqr=(x/sqr+sqr)/2.0;` на `sqr=(x/(sqr*sqr)+sqr)/2.0;`
2. Напишите программу для вычисления числа e (основание натуральных логарифмов) с заданной точностью, если известно, что
   $$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots \frac{1}{n!} + \ldots = 1 + \sum_{i=1}^{\infty} \frac{1}{i!}$$
   Для проверки: e≈2,718281828459045
3. Вычислите синус угла x (угол – в радианах), не используя функцию `sin` из библиотеки math.h, если известно, что:
   $$\sin x = \sum_{i=0}^{\infty} \frac{x^{2i+1}(-1)^i}{(2i+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots.$$
4. Вычислите косинус угла x (угол – в радианах), не используя функцию `cos` из библиотеки `math.h`, если известно, что:
   $$\cos x = 1 + \sum_{i=1}^{\infty} \frac{x^{2i}(-1)^i}{(2i)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots.$$

Примечание: для вычисления числа π, числа e, синусов и косинусов использованы ряды Тейлора. Подробнее о них можно узнать в учебниках по математическому анализу.