Информация для размещения на официальном сайте ГБПОУ

«Светлоградский региональный сельскохозяйственный колледж»

Для электронного обучения

Группа	421
Дата	08.11.2021
Время	8.10-13.00
Наименование УД/МДК/УП/ПП	Практические занятия УП-01
ФИО преподавателя	Коваленко Аркадий Владимирович
Электронная почта	aricus2007@inbox.ru
Основная литература	Программирование. Python. C++. Ч.1_Поляков К.Ю_2019
Тема	Классы исключений. Определение методов классов. Переопределение операций.
Задание	Практическая работа №12

Конструкция if

Конструкция **if** проверяет истинность условия, и если оно истинно, выполняет блок инструкций. Этот оператор имеет следующую сокращенную форму:

```
if (условие)
{
    инструкции;
}
```

В качестве *условия* использоваться условное выражение, которое возвращает **true** или **false**. Если условие возвращает true, то выполняются последующие инструкции, которые входят в блок if. Если условие возвращает false, то последующие инструкции не выполняются. Блок инструкций заключается в фигурные скобки.

Например:

```
#include <iostream>

int main()
{
    int x = 60;

    if(x > 50)
    {
```

```
9             std::cout << "x is greater than 50 \n";
10        }
11
12        if(x < 30)
13        {
14            std::cout << "x is less than 30 \n";
15        }
16
17        std::cout << "End of Program" << "\n";
18        return 0;
19    }
```

Здесь определены две условных конструкции if. Они проверят больше или меньше значение переменной x, чем определенное значение. В качестве инструкции в обоих случаях выполняется вывод некоторой строки на консоль.

В первом случае `x > 50` условие истинно, так как значение переменной x действительно больше 50, поэтому это условие возвратит true, и, следовательно, будут выполнятся инструкции, которые входят в блок if.

Во втором случае операция отношения `x < 30` возвратит false, так как условие ложно, поэтому последующий блок инструкций выполняться не будет. В итоге при запуске программы вывод консоли будет выглядеть следующим образом:

```
x greater than 50
End of Program
```

Также мы можем использовать полную форму конструкции if, которая включает оператор else:

```
1   if(выражение_условия)
2       инструкция_1
3   else
4       инструкция_2
```

После оператора else мы можем определить набор инструкций, которые выполняются, если условие в операторе if возвращает false. То есть если *условие* истинно, выполняются инструкции после оператора if, а если это выражение ложно, то выполняются инструкции после оператора else.

```
1   int x = 50;
2   if(x > 60)
3       std::cout << "x is greater than 60 \n";
4   else
```

```
5       std::cout << "x is less or equal 60 \n";
```

В данном случае условие `x > 60` ложно, то есть возвращает false, поэтому будет выполняться блок else. И в итоге на консоль будет выведена строка "x is less or equal 60 \n".

Однако нередко надо обработать не два возможных альтернативных варианта, а гораздо больше. Например, в случае выше можно насчитать три условия: переменная x может быть больше 60, меньше 60 и равна 60. Для проверки альтернативных условий мы можем вводить выражения **else if**:

```
1    int x = 60;
2
3    if(x > 60)
4    {
5        std::cout << "x is greater than 60 \n";
6    }
7    else if (x < 60)
8    {
9        std::cout << "x is less than 60 \n";
10   }
11   else
12   {
13       std::cout << "x is equal 60 \n";
14   }
```

То есть в данном случае мы получаем три ветки развития событий в программе.

Если в блоке if или else или else-if необходимо выполнить только одну инструкцию, то фигурные скобки можно опустить:

```
1   int x = 60;
2
3   if(x > 60)
4       std::cout << "x is greater than 60 \n";
5   else if (x < 60)
6       std::cout << "x is less than 60 \n";
7   else
8       std::cout << "x is equal 60 \n";
```

Конструкция switch

Другую форму организации ветвления программ представляет конструкция **switch...case**. Она имеет следующую форму:

```
1   switch(выражение)
2   {
```

```
3       case константа_1: инструкции_1;
4       case константа_2: инструкции_2;
5
6       default: инструкции;
7   }
```

После ключевого слова **switch** в скобках идет сравниваемое выражение. Значение этого выражения последовательно сравнивается со значениями после оператора **case**. И если совпадение будет найдено, то будет выполняться определенный блок **case**.

В конце конструкции switch может стоять блок **default**. Он необязателен и выполняется в том случае, если значение после switch не соответствует ни одному из операторов case. Например:

```
1    #include <iostream>
2
3    int main()
4    {
5        int x = 2;
6
7        switch(x)
8        {
9            case 1:
10               std::cout << "x = 1" << "\n";
11               break;
12           case 2:
13               std::cout << "x = 2" << "\n";
14               break;
15           case 3:
16               std::cout << "x = 3" << "\n";
17               break;
18           default:
19               std::cout << "x is undefined" << "\n";
20               break;
21       }
22       return 0;
23   }
```

Чтобы избежать выполнения последующих блоков case/default, в конце каждого блока ставится оператор **break**. То есть в данном случае будет выполняться оператор

```
1   case 2:
2       std::cout << "x = 2" << "\n";
3       break;
```

После выполнения оператора break произойдет выход из конструкции switch..case, и остальные операторы case будут проигнорированы. Поэтому на консоль будет выведена следующая строка

```
x = 2
```

Стоит отметить важность использования оператора break. Если мы его не укажем в блоке case, то после этого блока выполнение перейдет к следующему блоку case. Например, уберем из предыдущего примера все операторы break:

```
#include <iostream>

int main()
{
    int x = 2;

    switch(x)
    {
        case 1:
            std::cout << "x = 1" << "\n";
        case 2:
            std::cout << "x = 2" << "\n";
        case 3:
            std::cout << "x = 3" << "\n";
        default:
            std::cout << "x is undefined" << "\n";
    }
    return 0;
}
```

В этом случае опять же будет выполняться оператор `case 2:`, так как переменная x=2. Однако так как этот блок case не завершается оператором break, то после его завершения будет выполняться набор инструкций после `case 3:` даже несмотря на то, что переменная x по прежнему равна 2. В итоге мы получим следующий консольный вывод:

```
x = 2
x = 3
x is undefined
```

Тернарный оператор

Тернарный оператор **?:** позволяет сократить определение простейших условных конструкций if и имеет следующую форму:

```
[первый операнд - условие] ? [второй операнд] : [третий
```

Оператор использует сразу три операнда. В зависимости от условия тернарный оператор возвращает второй или третий операнд: если условие равно true (то есть истинно), то возвращается второй операнд; если условие равно false (то есть ложно), то третий. Например:

```
#include <iostream>

int main()
{
    setlocale(LC_ALL, "");
    int x = 5;
    int y = 3;
    char sign;
    std::cout << "Введите знак операции: ";
    std::cin >> sign;
    int result = sign=='+'?x + y:x - y;
    std::cout << "Результат: " << result << "\n";
    return 0;
}
```

В данном случае производится ввод знака операции. Здесь результатом тернарной операции является переменная result. И если переменная sign содержит знак "+", то result будет равно второму операнду - (x+y). Иначе result будет равно третьему операнду.

Задания для самостоятельного выполнения

№1

Конструкция if

```cpp
#include <iostream>

int main()
{
    int x = 60;

    if(x > 50)
    {
        std::cout << "x is greater than 50 \n";
    }

    if(x < 30)
    {
        std::cout << "x is less than 30 \n";
    }

    std::cout << "End of Program" << "\n";
    return 0;
}
```

Выполните программу и предоставьте скриншот

№2

Конструкция switch

```cpp
#include <iostream>

int main()
{
    int x = 2;

    switch(x)
    {
        case 1:
            std::cout << "x = 1" << "\n";
            break;
        case 2:
            std::cout << "x = 2" << "\n";
            break;
        case 3:
            std::cout << "x = 3" << "\n";
            break;
        default:
            std::cout << "x is undefined" << "\n";
            break;
    }
    return 0;
}
```

Выполните программу и предоставьте скриншот

№3

Тернарный оператор

```cpp
#include <iostream>

int main()
{
    setlocale(LC_ALL, "");
    int x = 5;
    int y = 3;
    char sign;
    std::cout << "Введите знак операции: ";
    std::cin >> sign;
    int result = sign=='+'?x + y:x - y;
    std::cout << "Результат: " << result << "\n";
    return 0;
}
```

Выполните программу и предоставьте скриншот

№4

Пользователь вводит число от 1 до 9999 (сумму выдачи в банкомате). Необходимо вывести на экран словами введенную сумму и в конце написать название валюты с правильным окончанием.

Выполните программу и предоставьте скриншот

№5

	Пользователь вводит порядковый номер пальца руки. Необходимо показать его название на экран. (с помощью оператора IF и switch) Выполните программу и предоставьте скриншот №6 Ввести три числа и найти наибольшее из них. Выполните программу и предоставьте скриншот №7 Определить является ли треугольник с заданными сторонами a,b,c равносторонним Выполните программу и предоставьте скриншот №8 Составить расписание на неделю. Пользователь вводит порядковый номер дня недели и у него на экране отображается, то, что запланировано на этот день. Выполните программу и предоставьте скриншот №9 Написать программу на C++ для вычисления(нахождения или решения) факториала Выполните программу и предоставьте скриншот №10 Напишите программу, которая вводит с клавиатуры три целых числа и записывает в логическую переменную значение `true` (истина), если среди введённых чисел есть хотя бы одно отрицательное. Выполните программу и предоставьте скриншот
Контрольный тест	Отправляйте отчет по Практической работе на электронную почту aricus2007@inbox.ru, название файла: ФИО студента и номер группы.

08.11.2021 А. В. Коваленко

Группа	421
Дата	12.11.2021
Время	8.10-13.00
Наименование УД/МДК/УП/ПП	Практические занятия УП-01
ФИО преподавателя	Коваленко Аркадий Владимирович
Электронная почта	aricus2007@inbox.ru
Основная литература	Программирование. Python. C++. Ч.1_Поляков К.Ю_2019
Тема	Основные операции с типами данных

Задание	Практическая работа №13
	Тема «Циклы»
	
	Для выполнения некоторых действий множество раз в зависимости от определенного условия используются циклы. В языке C++ имеются следующие виды циклов:
	
	• **for**
	• **while**
	• **do...while**
	
	Цикл while
	
	Цикл `while` выполняет некоторый код, пока его условие истинно, то есть возвращает true. Он имеет следующее формальное определение:
	
	`while(условие)`
	`{`
	`    // выполняемые действия`
	`}`
	
	После ключевого слова **while** в скобках идет условное выражение, которое возвращает true или false. Затем в фигурных скобках идет набор инструкций, которые составляют тело цикла. И пока условие возвращает true, будут выполняться инструкции в теле цикла.
	
	Например, выведем квадраты чисел от 1 до 9:
	
	`#include <iostream>`
	
	`int main()`
	`{`
	`    int i = 1;`
	`    while(i < 10)`
	`    {`

```
8           std::cout << i << " * " << i << " = " << i * i << std::e
9           i++;
10      }
11
12      return 0;
13  }
```

Здесь пока условие `i < 10` истинно, будет выполняться цикл while, в котором выводится на консоль квадрат числа и инкрементируется переменная i. В какой-то момент переменная i увеличится до 10, условие `i < 10` возвратит false, и цикл завершится.

Консольный вывод программы:

```
1 * 1 = 1
2 * 2 = 4
3 * 3 = 9
4 * 4 = 16
5 * 5 = 25
6 * 6 = 36
7 * 7 = 49
8 * 8 = 64
9 * 9 = 81
```

Каждый отдельный проход цикла называется итерацией. То есть в примере выше было 9 итераций.

Если цикл содержит одну инструкцию, то фигурные скобки можно опустить:

```
1  int i = 0;
2  while(++i < 10)
3      std::cout << i << " * " << i << " = " << i * i << std
```

Цикл for

Цикл for имеет следующее формальное определение:

```
1  for (выражение_1; выражение_2; выражение_3)
2  {
3      // тело цикла
4  }
```

выражение_1 выполняется один раз при начале выполнения цикла и представляет установку начальных условий, как

правило, это инициализация счетчиков - специальных переменных, которые используются для контроля за циклом.

выражение_2 представляет условие, при соблюдении которого выполняется цикл. Как правило, в качестве условия используется операция сравнения, и если она возвращает ненулевое значение (то есть условие истинно), то выполняется тело цикла, а затем вычисляется *выражение_3*.

выражение_3 задает изменение параметров цикла, нередко здесь происходит увеличение счетчиков цикла на единицу.

Например, перепишем программу по выводу квадратов чисел с помощью цикла for:

```
#include <iostream>

int main()
{
    for(int i =1; i < 10; i++)
    {
        std::cout << i << " * " << i << " = " << i * i <
    }

    return 0;
}
```

Первая часть объявления цикла - `int i = 1` - создает и инициализирует счетчик i. Фактически это то же самое, что и объявление и инициализация переменной. Счетчик необязательно должен представлять тип int. Это может быть и другой числовой тип, например, float. И перед выполнением цикла его значение будет равно 1.

Вторая часть - условие, при котором будет выполняться цикл. В данном случае цикл будет выполняться, пока переменная i не станет равна 10.

И третья часть - приращение счетчика на единицу. Опять же нам необязательно увеличивать на единицу. Можно уменьшать: i--. Можно изменять на другое значение: i+=2.

В итоге блок цикла сработает 9 раз, пока переменная i не станет равна 10. И каждый раз это значение будет увеличиваться на 1. И по сути мы получим тот же самый результат, что и в случае с циклом while:

```
1 * 1 = 1
2 * 2 = 4
3 * 3 = 9
4 * 4 = 16
5 * 5 = 25
6 * 6 = 36
7 * 7 = 49
8 * 8 = 64
9 * 9 = 81
```

Необязательно указывать все три выражения в определении цикла, мы можем одно или даже все из них опустить:

```
int i = 1;
for(; i < 10;)
{
    std::cout << i << " * " << i << " = " << i * i << std
    i++;
}
```

Формально определение цикла осталось тем же, только теперь первое и третье выражения в определении цикла отсутствуют: `for (; i < 10;)`. Переменная-счетчик определена и инициализирована вне цикла, а ее приращение происходит в самом цикле.

Можно определять вложенные циклы. Например, выведем таблицу умножения:

```
#include <iostream>

int main()
{
    for (int i=1; i < 10; i++)
    {
        for(int j = 1; j < 10; j++)
        {
            std::cout << i * j << "\t";
        }
        std::cout << std::endl;
    }

    return 0;
}
```

Цикл do

В цикле do сначала выполняется код цикла, а потом происходит проверка условия в инструкции while. И пока это условие истинно, то есть не равно 0, то цикл повторяется. Формальное определение цикла:

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

Например:

```
 #include <iostream>

 int main()
 {
     int i = 6;
     do
     {
         std::cout << i << std::endl;
         i--;
     }
     while(i>0);

     return 0;
 }
```

Здесь код цикла сработает 6 раз, пока i не станет равным нулю.

Но важно отметить, что цикл do гарантирует хотя бы однократное выполнение действий, даже если условие в инструкции while не будет истинно. То есть мы можем написать:

```
int i = -1;
do
{
    std::cout << i << std::endl;
    i--;
}
while(i>0);
}
```

Хотя у нас переменная i меньше 0, цикл все равно один раз выполнится.

Операторы continue и break

Иногда возникает необходимость выйти из цикла до его завершения. В этом случае можно воспользоваться оператором **break**. Например:

```
#include <iostream>

int main()
{
    int i = 1;
    for ( ; ; )
    {
        std::cout << i << " * " << i << " = " << i * i <
        i++;
        if (i > 9) break;
    }
    return 0;
}
```

Здесь когда значение переменной i достигнет 10, осуществляется выход из цикла с помощью оператора break.

В отличие от оператора break, оператор **continue** производит переход к следующей итерации. Например, нам надо посчитать сумму только нечетных чисел из некоторого диапазона:

```
#include <iostream>
int main()
{
    int result = 0;
    for (int i=0; i<10; i++)
    {
        if (i % 2 == 0) continue;
        result +=i;
    }
    std::cout << "result = " << result << std::endl; //
    return 0;
}
```

Чтобы узнать, четное ли число, мы получаем остаток от целочисленного деления на 2, и если он равен 0, то с помощью оператора continue переходим к следующей итерации цикла. А если число нечетное, то складываем его с остальными нечетными числами.

Задание №1
Цикл while

```cpp
#include <iostream>

int main()
{
    int i = 1;
    while(i < 10)
    {
        std::cout << i << " * " << i << " = " << i * i << std::endl;
        i++;
    }

    return 0;
}
```

Выполнить и предоставить скриншот

Задание №2
Цикл for

```cpp
#include <iostream>

int main()
{
    for(int i =1; i < 10; i++)
    {
        std::cout << i << " * " << i << " = " << i * i << std::endl;
    }

    return 0;
}
```

Выполнить и предоставить скриншот
Задание №3
Вложенный цикл

```cpp
#include <iostream>

int main()
{
    for (int i=1; i < 10; i++)
    {
        for(int j = 1; j < 10; j++)
        {
            std::cout << i * j << "\t";
        }
        std::cout << std::endl;
    }

    return 0;
}
```

Выполнить и предоставить скриншот

Задание №4
Цикл do

```cpp
#include <iostream>

int main()
{
    int i = 6;
    do
    {
        std::cout << i << std::endl;
        i--;
    }
    while(i>0);

    return 0;
}
```

Выполнить и предоставить скриншот

Задание №5

Операторы continue и break

```cpp
#include <iostream>

int main()
{
    int i = 1;
    for ( ; ; )
    {
        std::cout << i << " * " << i << " = " << i * i << std::endl;
        i++;
        if (i > 9) break;
    }
    return 0;
}
```

```cpp
#include <iostream>
int main()
{
    int result = 0;
    for (int i=0; i<10; i++)
    {
        if (i % 2 == 0) continue;
        result +=i;
    }
    std::cout << "result = " << result << std::endl; // 25
    return 0;
}
```

Выполнить и предоставить скриншот

Задание №6

Составьте программу, выводящую на экран квадраты чисел от 10 до 20 включительно.

Выполнить и предоставить код программы

Задание №7

Даны натуральные числа от 35 до 87. Вывести на консоль те из них, которые при делении на 7 дают остаток 1, 2 или 5.

Выполнить и предоставить код программы

	Задание №8 **Найдите сумму** 1+2+3+…+n**, где число** n **вводится пользователем с клавиатуры.** Выполнить и предоставить код программы Задание №9 **Найдите произведение цифр трехзначного числа** Выполнить и предоставить код программы Задание №10 **Найдите наибольшую цифру данного натурального числа.** Выполнить и предоставить код программы
Контрольные ответы	Отправляйте отчет по Практической работе на электронную почту aricus2007@inbox.ru, название файла: ФИО студента и номер группы.

12.11.2021 А. В. Коваленко

Группа	421
Дата	13.11.2021
Время	8.10-13.00
Наименование УД/МДК/УП/ПП	Практические занятия УП-01
ФИО преподавателя	Коваленко Аркадий Владимирович
Электронная почта	aricus2007@inbox.ru
Основная литература	Программирование. Python. C++. Ч.2_Поляков К.Ю_2019
Тема	Выполнение тестирования программных модулей с помощью массивов данных

Задания	**Массивы**
	
	Массив представляет набор однотипных данных. Формальное определение массива выглядит следующим образом:
	
	`тип_переменной название_массива [длина_массива]`
	
	После типа переменной идет название массива, а затем в квадратных скобках его размер. Например, определим массив из 4 чисел:
	
	`int numbers[4];`
	
	Данный массив имеет четыре числа, но все эти числа имеют неопределенное значение. Однако мы можем выполнить инициализацию и присвоить этим числам некоторые начальные значения через фигурные скобки:
	
	`int numbers[4] = {1,2,3,4};`
	
	Значения в фигурных скобках еще называют инициализаторами. Если инициализаторов меньше, чем элементов в массиве, то инициализаторы используются для первых элементов. Если в инициализаторов больше, чем элементов в массиве, то при компиляции возникнет ошибка:
	
	`int numbers[4] = {1, 2, 3, 4, 5, 6};`
	
	Здесь массив имеет размер 4, однако ему передается 6 значений.
	
	Если размер массива не указан явно, то он выводится из количества инициализаторов:
	
	`int numbers[] = {1, 2, 3, 4, 5, 6};`

В данном случае в массиве есть 6 элементов.

Свои особенности имеет инициализация символьных массивов. Мы можем передать символьному массиву как набор инициализаторов, так и строку:

```
char s1[] = {'h', 'e', 'l', 'l', 'o'};
char s2[] = "world";
```

Причем во втором случае массив s2 будет иметь не 5 элементов, а 6, поскольку при инициализации строкой в символьный массив автоматически добавляется нулевой символ '\0'.

При этом не допускается присвоение одному массиву другого массива:

```
int nums1[] = {1,2,3,4,5};
int nums2[] = nums1;     // ошибка
nums2 = nums1;           // ошибка
```

После определения массива мы можем обратиться к его отдельным элементам по индексу. Индексы начинаются с нуля, поэтому для обращения к первому элементу необходимо использовать индекс 0. Обратившись к элементу по индексу, мы можем получить его значение, либо изменить его:

```
#include <iostream>

int main()
{
    int numbers[4] = {1,2,3,4};
    int first_number = numbers[0];
    std::cout << first_number << std::endl; // 1
    numbers[0] = 34;                        // изменяем
    std::cout << numbers[0] << std::endl; // 34

    return 0;
}
```

Число элементов массива также можно определять через константу:

```
const int n = 4;
int numbers[n] = {1,2,3,4};
```

Перебор массивов

Используя циклы, можно пробежаться по всему массиву и через индексы обратиться к его элементам:

```
#include <iostream>

int main()
{
    int numbers[4] = {1,2,3,4};
    int size = sizeof(numbers)/sizeof(numbers[0]);
    for(int i=0; i < size; i++)
        std::cout << numbers[i] << std::endl;

    return 0;
}
```

Чтобы пройтись по массиву в цикле, вначале надо найти длину массива. Для нахождения длины применяется оператор **sizeof**. По сути длина массива равна совокупной длине его элементов. Все элементы представляют один и тот же тип и занимают один и тот же размер в памяти. Поэтому с помощью выражения `sizeof(numbers)` находим длину всего массива в байтах, а с помощью выражения `sizeof(numbers[0])` - длину одного элемента в байтах. Разделив два значения, можно получить количество элементов в массиве. А далее с помощью цикла for перебираем все элементы, пока счетчик i не станет равным длине массива. В итоге на консоль будут выведены все элементы массива:

```
1
2
3
4
```

Но также есть и еще одна форма цикла **for**, которая предназначена специально для работа с коллекциями, в том числе с массивами. Эта форма имеет следующее формальное определение:

```
for(тип переменная : коллекция)
{
    инструкции;
}
```

Используем эту форму для перебора массива:

```
#include <iostream>

```

	3 int main() 4 { 5 int numbers[4] = {1,2,3,4}; 6 for(int number : numbers) 7 std::cout << number << std::endl; 8 9 return 0; 10 } При переборе массива каждый перебираемый элемент будет помещаться в переменную number, значение которой в цикле выводится на консоль. Задание №1 **Удалить в массиве все числа, которые повторяются более двух раз.** Выполнить и предоставить код программы Задание №2 **Введите одномерный целочисленный массив. Найдите наибольший нечетный элемент. Далее трижды осуществите циклический сдвиг влево элементов, стоящих справа от найденного максимума, и один раз сдвиг элементов вправо, стоящих слева от найденного максимума.** Выполнить и предоставить код программы Задание №3 **Найдите сумму отрицательных элементов массива.** Выполнить и предоставить код программы
Контроль	Отправляйте отчет по Практической работе на электронную почту aricus2007@inbox.ru, название файла: ФИО студента и номер группы.

13.11.2021 А. В. Коваленко