РЕШИТЬ ПРАКТИЧЕСКИЕ ЗАДАЧИ НА C++,

используя циклы, условия, функции и классы

РАБИН АЛЕКСЕЙ ВЛАДИМИРОВИЧ

д-р техн. наук, директор центра координации научных исследований,
Санкт-Петербургский государственный университет аэрокосмического приборостроения (ГУАП)

КОНСТРУКЦИЯ IF

Конструкция **if** проверяет истинность условия: если оно истинно, выполняет блок инструкций

Этот оператор имеет следующую сокращенную форму:

```
1    if (условие)
2    {
3        инструкции;
4    }
```

Например:

```
1    #include <iostream>
2     int main()
3    {
4        int x = 60;
5
6        if(x > 50)
7        {
8            std::cout << "x is greater than 50 \n";
9        }
10
11       if(x < 30)
12       {
13           std::cout << "x is less than 30 \n";
14       }
15
16       std::cout << "End of Program" << "\n";
17       return 0;
18   }
```

КОНСТРУКЦИЯ IF

Полная форма конструкции **if**

```
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**

Если *условие* истинно, выполняются инструкции после оператора **if**, 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";
```

КОНСТРУКЦИЯ IF

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   }
```

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

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

КОНСТРУКЦИЯ SWITCH

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

```
switch(выражение)
{
    case константа_1: инструкции_1;
    case константа_2: инструкции_2;

    default: инструкции;
}
```

Например

```
#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;
}
```

КОНСТРУКЦИЯ SWITCH

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:

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

ТЕРНАРНЫЙ ОПЕРАТОР

Тернарный оператор

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

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.

```
1    #include <iostream>
2     int main()
3    {
4        setlocale(LC_ALL, "");
5        int x = 5;
6        int y = 3;
7        char sign;
8        std::cout << "Введите знак операции: ";
9        std::cin >> sign;
10       int result = sign=='+'?x + y:x - y;
11       std::cout << "Результат: " << result << "\n";
12       return 0;
13   }
```

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
}
```

Например, выведем квадраты чисел **от 1 до 9:**

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

    return 0;
}
```

КОНСОЛЬНЫЙ ВЫВОД ПРОГРАММЫ ПРИМЕРА:

```
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
```

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: 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

ЦИКЛ FOR

Программу по выводу квадратов чисел
с помощью цикла for:

```
1    #include <iostream>
2
3    int main()
4    {
5        for(int i =1; i < 10; i++)
6        {
7            std::cout << i << " * " << i << " = " << i * i << std::endl;
8        }
9
10       return 0;
11   }
```

Консольный вывод программы примера:

```
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
```

ЦИКЛ FOR

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

```
1    int i = 1;
2    for(; i < 10;)
3    {
4        std::cout << i << " * " << i << " = " << i * i << std::endl;
5        i++;
6    }
```

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

Консольный вывод программы примера:

```
1    #include <iostream>
2
3    int main()
4    {
5        for (int i=1; i < 10; i++)
6        {
7            for(int j = 1; j < 10; j++)
8            {
9                std::cout << i * j << "\t";
10           }
11           std::cout << std::endl;
12       }
13
14       return 0;
15   }
```

ЦИКЛ DO...WHILE

В цикле **do...while** сначала выполняется код цикла, а потом происходит проверка условия в инструкции **while**

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;
}
```

ЦИКЛ DO...WHILE

Важно отметить, что цикл **do** гарантирует хотя бы однократное выполнение действий, даже если условие в инструкции **while** не будет истинно

Мы можем написать:

```
1    int i = -1;
2    do
3    {
4        std::cout << i << std::endl;
5        i--;
6    }
7    while(i>0);
8    }
```

ОПЕРАТОРЫ CONTINUE И BREAK

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

Когда значение переменной i достигнет 10, осуществляется выход из цикла с помощью оператора **break**

Например:

```
1    #include <iostream>
2     int main()
3    {
4       int i = 1;
5       for ( ; ; )
6       {
7          std::cout << i << " * " << i << " = " << i * i << std::endl;
8          i++;
9          if (i > 9) break;
10      }
11      return 0;
12   }
```

ОПЕРАТОРЫ CONTINUE И BREAK

В отличие от оператора **break,**
оператор **continue** производит переход
к следующей итерации

Например, нам надо посчитать сумму только
нечетных чисел из некоторого диапазона:

```
1   #include <iostream>
2   int main()
3   {
4       int result = 0;
5       for (int i=0; i<10; i++)
6       {
7           if (i % 2 == 0) continue;
8           result +=i;
9       }
10      std::cout << "result = " << result << std::endl; // 25
11      return 0;
12  }
```

ФУНКЦИИ C++

Задания для примера решения:

Объявить два целочисленных массива
с разными размерами и написать функцию,
которая заполняет их элементы значениями
и показывает на экран.

Функция должна принимать два параметра – массив и его размер.

```
#include <iostream>
using namespace std;
void fillAndShowArray(int arrayForFilling[], int size);
int main()
{
	const int SIZE1 = 8;
	const int SIZE2 = 14;

	int arrayForFilling1 [SIZE1] = {};
	int arrayForFilling2 [SIZE2] = {};

	fillAndShowArray(arrayForFilling1, SIZE1);
	fillAndShowArray(arrayForFilling2, SIZE2);

	return 0;
}

void fillAndShowArray(int arrayForFilling[], int size)
{
	for (int i = 0; i < size; i++)
	{
		arrayForFilling[i] = i + 1;
		cout << arrayForFilling[i] << "  ";
	}
	cout << endl;
}
```

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Определение классов

Класс может определять переменные и константы для хранения состояния объекта и функции для определения поведения объекта.

Например, определим простейший класс:

```
class Person
{

};
```

Для определения класса применяется ключевое слово **class**, после которого идет собственно название класса. В данном случае класс называется **Person** и представляет человека. После названия класса идет блок кода, который определяет тело класса.

После определения класса мы можем создавать его переменные:

```
class Person
{

};
int main()
{
    Person person;
    return 0;
}
```

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Теперь класс **Person** имеет две переменных **name** и **age**, которые предназначены для хранения имени и возраста человека соответственно

Также класс определяет функцию **move**, которая выводит строку на консоль

Стоит обратить внимание на модификатор доступа **public**:, который указывает, что идущие после него переменные и функции будут доступны извне, из внешнего кода

Теперь изменим класс

```
#include <iostream>
#include <string>
using std::string;
using std::cout;
using std::endl;

class Person
{
public:
    string name;
    int age;
    void move() {
        cout << name << " is moving"<< endl;
    }
};
int main()
{
    Person person;
    person.name = "Tom";
    person.age = 22;
    cout << "Name: " << person.name << "\tAge: " << person.age << endl;
    person.move();

    return 0;
}
```

УКАЗАТЕЛИ НА ОБЪЕКТЫ КЛАССОВ

На объекты классов, как и на объекты других типов, можно определять указатели

Затем через указатель можно обращаться к членам класса - переменным и методам

Однако если при обращении через обычную переменную используется символ точка,
то для обращения к членам класса через указатель применяется стрелка (->):

```
1   #include <iostream>
2   #include <string>
3   using std::string;
4   using std::cout;
5   using std::endl;
6
7   class Person
8   {
9   public:
10      string name;
11      int age;
12      void move() {
13          cout << name << " is moving" << endl;
14      }
15  };
16  int main()
17  {
18      Person person;
19      Person *ptr = &person;
20      ptr->name = "Tom";
21      ptr->age = 22;
22      ptr->move();
23      cout << "Name: " << ptr->name << "\tAge: " << ptr->age << endl;
24      cout << "Name: " << person.name << "\tAge: " << person.age << endl;
25
26      return 0;
27  }
```

СПАСИБО ЗА ВНИМАНИЕ