Практические занятия с применением C++

Примеры и задания выполняются в Online-компиляторе, поддерживающий язык программирования C++:

http://cpp.sh/

http://www.pythontutor.com/visualize.html#mode=edit

http://rextester.com/l/cpp_online_compiler_visual

Занятие 2. Условные конструкции

Цель: познакомиться с реализацией ветвления в алгоритмах выбора.

Задание 1. Задача выбора (поиска)

Дана точка в декартовой системе координат P(x,y). Требуется составить условие определения в какой четверти находится данная точка.

- Ознакомьтесь с вариантами алгоритмического решения задачи.

Вариант 1. Последовательный перебор. Последовательно составляются условия на принадлежность к каждой четверти.

Вариант 2. Метод деления пополам. Сначала сравнивается первая координата на условие принадлежности точки к половине системы координат. Затем уточняется к какой из ее двух частей принадлежит точка.

Для решения задач выбора применяются:

- Операции сравнения, которые возвращают True (истина) или False (ложь):
  - меньше, больше, равно, не равно, больше или равно, меньше или равно
- Логические операторы, применяемые для проверки одновременно несколько условий:
  - `X and Y` (Истина, если оба значения X и Y истинны)
  - `X or Y` (Истина, если хотя бы одно из значений X или Y истинно)
  - `not X` (Истина, если X ложно).

- Программное решение. Введите текст программ по каждому варианту. Сравните реализации между собой, какая вам кажется более удобной?

Вариант 1. Последовательный перебор. Реализуется цепочкой операторов if-else.

```
#include <iostream>
using namespace std;


  int main()
  {
```

```
    double x, y;
    cin >> x;
    cin >> y;

    if (x > 0 && y > 0)
        cout << "I qv" << endl;
    else if (x > 0 && y < 0)
        cout << "IV qv" << endl;
    else if (y > 0)
        cout << "II qv" << endl;
    else
        cout << "III qv" << endl;

    return 0;
}
```

Вариант 2. Метод деления пополам. Реализуется вложенной конструкцией операторов выбора.

```
int main()
{
   double x, y;
   cin >> x;
   cin >> y;
    if (x > 0)
    {
        if (y > 0)                  // x>0, y>0
        {
            cout << "I qv" << endl;
        }
        else                        // x>0, y<0
        {
            cout << "IV qv" << endl;
        }
    }
    else
    {
        if (y > 0)                  // x<0, y>0
        {
            cout << "II qv" << endl;
        }
        else                        // x<0, y<0
        {
```

```
            cout << "III qv " << endl;
        }
    }
  return 0;
  }
```

Задание 2. Множественный выбор типа операции

Требуется реализовать калькулятор, выполняющий набор из четырех операций: сложение, вычитание, умножение и деление.

Выбор комплектации

- В функции `main()` объявите три переменные вещественного типа (два операнда и результат) и переменную символьного типа, которая будет определять выбор пользователя:

```
#include <iostream>
using namespace std;

int main()
{
     double x, y, z;
     char op;
```

- Реализуйте запрос от пользователя для ввода операндов и символа операции:

```
     cout << "x: ";
     cin >> x;
     cout << "operation:  ";
     cin >> op;
     cout << "y:  ";
     cin >> y;
```

- Реализуйте выбор альтернативного варианта с использованием оператора `switch`:

```
        switch (op)
        {
        case '+':
                  z = x+y;
                  break;
        case '-':
                  z = x-y;
                  break;
        case '*':
                  z = x*y;
                  break;
        case '/':
                  z = x/y;
                  break;
        }
     cout << "z = " << z << endl;
return 0;
```

```
}
```

- Запустите программу, проверьте ее работу. Попробуйте ввести для второго операнда значение нуля. Посмотрите реакцию программы.
- Операцию деления можно указать и символом «двоеточие», добавьте в case-ветвь `case '/'` эту возможность:

```
case '/':
case ':':
            z = x/y;
            break;
```

- Запустите программу, проверьте ее работу.

Занятие 3. Циклы

Цель: познакомиться с реализацией циклов в алгоритмах, реализующих повторяющиеся действия.

Задание 1. Определение наибольшего общего делителя. Цикл с предусловием (while)

Универсальным способом, позволяющим вычислять наибольший общий делитель (наибольший общий делитель (НОД) – наибольшее число, на которое можно разделить несколько чисел без остатка) двух положительных целых чисел является алгоритм Евклида.

Алгоритм Евклида с вычитанием (один из возможных вариантов реализации) заключается в последовательной замене наибольшего числа из двух данных чисел, для которых вычисляется НОД, разностью этих чисел.

Например, НОД(24,18) = НОД(6,18) = НОД(6,12) = НОД(6,6). В результате наибольший общий делитель чисел 24 и 18 равен 6.

- В функции `main()` объявите две переменные целого типа и реализуйте их ввод с клавиатуры:

```
#include <iostream>
using namespace std;

int main()
{
      int a, b;
      cout << "a = ";
      cin >> a;
      cout << "b = ";
      cin >> b;
```

- Алгоритм Евклида с вычитанием реализован с помощью цикла с предусловием (while) – тело цикла выполняется *пока условие цикла истинно*. Добавьте далее в функцию main() следующий код этого цикла:

```
while (a != b)
{
```

```
            if (a > b)
                  a -= b; // аналогично выражению a = a - b
            else
                  b -= a;
      }

      cout << a << endl;
      return 0;
}
```

- Запустите программу, проверьте работу алгоритма.

Задание 2. Применение цикла с параметром (for)

Цикл с параметром применяется в случае, когда известно количество итераций – повторений некоторых действий – тела цикла.

Требуется составить программу тестирования знаний таблицы умножения. Тест должен иметь заранее известное число вопросов и подсчитывать количество неправильных ответов.

Для генерации случайных множителей будет использована функция rand(), а для привязки процесса генерации к системному времени (чтобы при повторном запуске числа были другие) – функция srand(time(NULL)).

- В функции `main()` объявите три переменные целого типа: два операнда (a, b) и результат умножения (c)::

```
#include <iostream>
#include <ctime>

using namespace std;

int main()
{
      srand(time(NULL));
      int a, b, c;
```

- Далее объявите еще две переменных целого типа: для подсчета неправильных ответов (k = 0) и общего числа вопросов (n = 10):

```
      int k = 0, n = 10;
```

- Добавьте реализацию цикла for, изучите содержимое тела цикла:

```
  for(int i = 1;i <= n ;i++)
  {
     // инициализация операндов случайными числами от 1 до 10[1]
        a = rand() % 10 + 1;
        b = rand() % 10 + 1;

        cout << a << " * " << b << " = ";
        cin >> c;
```

[1] формула для диапазона: r = min + rand() % (max – min +1)

```
        if (a*b != c)
        {
              k++; // операция «инкремент», аналогично: k = k + 1
              cout << "Error! ";
              cout << a << " * " << b << " = " << a * b << endl;
        }
   }

  cout << "Count error: " << k << endl;
  return 0;

}
```

- Запустите программу, проверьте работу алгоритма.

Задание 3. Определение значение функции на заданном интервале. Цикл с постусловием (do while)

Требуется разработать программу вычисления значений аргумента некоторой функции на указанном интервале. При фиксированной левой границе целесообразнее использовать цикл с постусловием*: выполняет тело цикла до тех пор, пока условие цикла истинно*.

- В функции `main()` объявите две переменные целого типа: первая определяет значение левой границы интервала, вторая – значение функции:

```
#include <iostream>
#include <math.h>

using namespace std;

int main()
{
      double x = 0, y;
```

- Укажите цикл с постусловием, в теле цикла рассчитайте значение аргумента функции для текущей точки и выведите на экран значения функции для этой точки:

```
      do
      {
            y = sin(x);
            cout <<"\t"<< x <<"\t"<< y << endl;
            x = x + 0.01;                    // шаг цикла
      }
      while (x <= 3.14/2);
      return 0;
}
```

- Запустите программу, проверьте вывод расчетных данных.

Задание 4. Управление циклом – операторы break и continue

Оператор `break` досрочно прерывает цикл

- В пример с циклом for добавьте в ветвь проверки ответа if еще одну проверку: проверку на допустимое число допущенных ошибок: если их число становится больше 2, то тест прекращается (выделено жирным):

```
for(int i=1; i<=n ;i++)
 {
      a = rand() % 10 + 1;
      b = rand() % 10 + 1;

      cout << a << " * " << b << " = ";
      cin >> c;

      if (a*b != c)
      {
        k++;
        if (k>2)
        {
            cout << "Stop test!";
            break;
        }
          cout << "Error!";
          cout << a << " * " << b << " = " << a*b << endl;
      }
 }
```

- Запустите программу, проверьте ее работу – после двух ошибок тест должен прекратиться.

Оператор `continue` начинает следующий проход цикла, минуя оставшееся тело цикла

- В пример с циклом for после ввода данных добавьте новую проверку вводимого числа: если выпадет множитель 10 (или a или b), то дальнейшие проверки должны быть пропущены и счетчик попыток не должен сработать, ведь задание получается слишком легким (выделено жирным):

```
for(int i=1; i<=n ;i++)
 {
      a = rand() % 10 + 1;
      b = rand() % 10 + 1;

      cout << a << " * " << b << " = ";
      cin >> c;

      if (a == 10 || b == 10)
      {
            cout << "Easy! Repeat\n";
            continue;
      }
      if (a*b != c)
      {
```

```
            k++;
          if (k>2)
          {
              cout << "Stop test! ";
              break;
          }
            cout << "Error! ";
            cout << a << " * " << b << " = " << a*b << endl;
       }
  }
```

- Запустите программу, проверьте ее работу – при появлении в качестве проверочных чисел десяток – проверка должна быть проигнорирована и попытка не засчитается.

Занятие 4. Применение структур данных

Цель: познакомиться с организацией данных в виде структур данных на примере массива и вектора.

Задание 1. Сохранение результатов в массиве

В программе тестирования знаний таблицы умножения (Занятие 3, задание 2) требуется сохранять введенные ответы и затем выводить результаты на экран.

- Подумайте, в какой структуре данных можно сохранять вводимые ответы?

Ход рассуждений:

- Данные одного типа?
- Да, планируется сохранять значения переменой *c*, а она целого типа.
- Заранее известно сколько должно храниться значений?
- Да, по замыслу теста предполагается 10 вопросов.
- Есть ли особые требования к размещению данных в памяти?
- Нет, рекомендуется использовать наиболее удобный и быстрый способ хранения и обработки данных.

Итог рассуждений: наиболее подходящая структура данных – массив.

- В функции main() программы тестирования знаний таблицы умножения (Занятие 3, задание 2) замените объявление переменной *n* на объявление константы и далее объявите массив для хранения десяти целых чисел:

```
int main()
{
      srand(time(NULL));
      int a, b, c;
      int k = 0;

      const int n = 10;
      int mas[n];
```

- В заголовок цикла for внесите изменения: вместо `int i=1; i<=n` укажите индексацию с нуля (так как массивы индексируются с нулевого индекса) и условие цикла не включает n: `int i=0; i<n`

```
for(int i=0; i<n ;i++)
{
```

- Далее в теле цикла после ввода переменной *c* (ответа на тест) добавьте выражение, присваивающее текущему элементу массива значение этой переменной c:

```
cin >> c;
mas[i] = c;
```

- В функции main() перед выводом количества ошибок создайте новый цикл for для вывода на экран всех введенных ответов:

```
cout << "\nAll: ";
for (int i=0; i<n; i++)
{
      cout << mas[i] << ends;
}
```

- Запустите программу, проверьте ее работу, должны быть выведены элементы массива – введенные вами ответы.

Задание 2. Сохранение набора результатов неизвестного размера

Теперь в программе тестирования знаний таблицы умножения требуется также сохранять введенные правильные и не правильные ответы и затем выводить результаты на экран.

- Подумайте, в какой структуре данных можно сохранять правильные и не правильные ответы?

Ход рассуждений:

- Данные одного типа?
- Да, планируется сохранять значения переменой *c*, а она целого типа.
- Заранее известно сколько должно храниться значений?
- Нет, по замыслу теста предполагается 10 вопросов, сколько из них правильных и не правильных не известно.
- Есть ли особые требования к размещению данных в памяти?
- Нет, рекомендуется использовать наиболее удобный и быстрый способ хранения и обработки данных.

Итог рассуждений: наиболее подходящая структура данных – вектор – динамическая структура, позволяющая доступ к элементу по индексу.

- В функции main() после объявления массива объявите два вектора (они изначально пустые) для хранения целых чисел:

```
vector<int> v1;
vector<int> v2;
```

- В цикле for в конструкции if укажите код для добавления неправильного введенного ответа в вектор v2:

```
if (a*b != c)
{
      v2.push_back(c);
      k++;
```

- После блока if добавьте блок else для реализации ветви в случае введения правильного ответа и в этом блоке укажите код для добавления правильного введенного ответа в вектор v1:

```
else
{
      v1.push_back(c);
}
```

- После цикла выводящего все значения массива добавьте два цикла for для аналогичного вывода правильных и неправильных ответов (для определения условия продолжения цикла примените для вектора стандартную функцию size()):

```
cout << "\nGood: ";
for (int i = 0; i < v1.size(); i++)
{
      cout << v1[i] << ends;
}

cout << "\nBad: ";
for (int i = 0; i < v2.size(); i++)
{
      cout << v2[i] << ends;
}
```

- Запустите программу, проверьте ее работу, должны быть выведены элементы массива – введенные вами ответы и элементы векторов – правильные и неправильные ответы.