Практические занятия с применением C++

Online-компиляторы, поддерживающий язык программирования C++:

http://cpp.sh/

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

http://rextester.com/l/cpp_online_compiler_visual

Занятие 5. Использование функций

На этом занятии демонстрируется организация программ в соответствии с процедурным стилем программирования, правила описания, объявления и вызова функций.

Вспомните, на первом занятии вы знакомились со следующей программой:

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

using namespace std;
int main()
{
    double p;
    cout << "Input p:\n";
    cin >> p;
// вычисление стороны
    double st = p/3;
// вычисление площади
    double s = sqrt(p/2*(p/2-st)*(p/2-st)*(p/2-st));
    cout << "s = " << s << endl;
    return 0;
}
```

- Обратите внимание, что в функции main() выполняются действия различные по своей «природе»: ввод и вывод данных на консоль, а также арифметические расчеты. Есть смысл разделить весь функционал на отдельные части (модули). В процедурном программировании отдельным модулем являются функция и процедура.

Задание 1. Реализация алгоритма вычисления площади в отдельной функции

Объявление функции

- Объявите в начале программы до функции `main()` функцию, которая будет принимать один параметр вещественного типа – периметр и возвращать значение площади:

```
double Sq(double p)
{
    double st = p/3;
    double s = sqrt(p/2*(p/2-st)*(p/2-st)*(p/2-st));
    return s;
}
```

Вызов функции

- В функции `main()` вместо непосредственного расчета вызовите созданную функцию с передачей ей в качестве параметра значения периметра (выделено красным шрифтом) и присвойте результат другой переменной:

```
int main()
{
    double p;
    cout << "Input p:\n";
    cin >> p;
    double s = Sq(p);
    cout << "s = " << s << endl;

return 0;
}
```

- Запустите программу, проверьте ее работу.

Использование прототипа

- Перенесите объявление функции после функции main(), а перед функцией main() добавьте прототип этой функции:

```
double Sq(double);
```

- Запустите программу, проверьте ее работу.

Задание 2. Реализация взаимодействия пользователя с программой

Для полного разделения программы на модули осталось выделить алгоритмы ввода с клавиатуры периметра и вывода площади на экран в отдельные модули.

- Объявите в начале программы до функции `main()` функцию, которая не будет принимать параметры, но должна возвращать значение введенного периметра:

```
double Input()
{
   double p;
   cout << "Input p:\n";
   cin >> p;
   return p;
}
```

- В функции `main()` вместо непосредственного запроса и ввода периметра укажите вызов созданной функции:

```
int main()
{
    double p = Input();
    double s = Sq(p);
    ...
```

- Запустите программу, проверьте ее работу.
- Объявите в начале программы до функции `main()` функцию, которая будет принимать параметр – значение площади, но не будет ничего возвращать, т.е. просто отрабатывать указанные в ее теле инструкции (формально такую функцию можно назвать процедурой):

```
void Output(double s)
{
    cout << "s = " << s << endl;
}
```

- Обратите внимание на ключевое слово `void` – оно означает, что это функция (по сути процедура) ничего не возвращает.
- В функции `main()` вместо непосредственного вывода укажите вызов созданной функции с передачей ей полученной от функции расчета значение площади:

```
int main()
{
    double p = Input();
    double s = Sq(p);
    Output(s);
    return 0;
}
```

- Запустите программу, проверьте ее работу.

Замечание. Функции ввода и вывода тоже можно было бы реализовать с помощью прототипов.

- Обратите внимание, что теперь функция `main()` только организует последовательность вызова необходимых функций для решения главной задачи программы.

Занятие 6. Понятие указателей и ссылок

На этом занятии рассматриваются указатели и ссылки и демонстрируется их применение.

Указатели в С++ представляют собой вид переменных и хранят адрес памяти, по которой размещена другая переменная.

Задание 1. Объявление указателя и его использование для доступа к ячейке памяти

- В функции main() объявите переменную и указатель на нее:

```
#include <iostream>
using namespace std;
int main()
{
   int a = 10;
   int *pa;        // Объявили указатель на целое
   pa = &a;       // Инициализировали указатель адресом переменной  a
                    // (& - операция получения адреса)
```

- Затем выведете на экран значение переменной-указателя и значение адреса этой переменной указателя:

```
 cout << pa << endl; // Что будет если вывести значение указателя?
                          // будет адрес, хранимый в указателе
 cout << &pa << endl;    // Что будет если вывести значение указателя
                          // с помощью оператора &, чей это адрес?
```

- С помощью оператора разыменовывания (символ *) извлеките хранящееся в ячейке значение и прибавьте к нему другое значение, например:

```
  *pa += 15;                //Ко значению, хранящимся в адресе
             // памяти, полученном из указателя будет прибавлено 15
```

- Теперь выведете на экран значение переменной-указателя и значение первоначальной переменной для сравнения результата (убедитесь, что это одна и та же ячейка памяти) указателя:

```
  cout << *pa << endl;    // по адресу, хранимому в указателе
     // выводим значение, хранимое в памяти на экран.
  cout << a << endl;    // а что теперь хранится в переменной а?


  return 0;
}
```

Задание 2. Использование указателей при работе с массивами

Имя массива является адресом его первого элемента. Соответственно с помощью операции разыменования можно получить значение по этому адресу. Например, если *mass* – имя массива, то адрес второго элемента будет представлять выражение *mass+1*, а его значение **(mass+1)*.

- Введите текст следующей программы, обратите внимание на реализацию доступа к элементам массива в первом цикле – использование операции разыменовывания (*) для ввода значений элементов при перемещении по

массиву, а также во втором цикле – для получения значений элементов (выделено красным шрифтом):

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

int main()
{
    int myArray[5];
    for (int i=0; i<5; i++)
    {
        cout << "Value for myArray: ";
        cin >> *(myArray + i);

    }
    for (int i = 0; i<5; i++)
        cout << i << ": " << *(myArray + i) << "\n";
    return 0;
}
```

- Проверьте, как можно использовать указатели для перемещения по массиву, для этого в первом цикле замените итерационную переменную i на указатель (изменения красным шрифтом):

```
int main()
{
    int myArray[5];
    int *pAr = myArray;
    for (pAr; pAr<&myArray[5]; pAr++)
    {
        cout << "Value for myArray: ";
        cin >> *(pAr);
    }

    for (int i = 0; i<5; i++)
        cout << i << ": " << *(myArray + i) << "\n";
    return 0;
}
```

Задание 3. Исследование ссылок

Ссылка – это другое имя уже существующего объекта, т.е. это еще одно имя переменной – псевдоним.

Ссылка описывается как переменная и перед ее объявлением обязательно объявляют объект, на который она указывает.

- Введите следующую программу.
- Запустите программу, проверьте значения переменных.

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

int main()
{
   int x = 10;
   int y = 20;
   int& rx = x; // объявлена ссылка на x
   rx += y;
   rx = rx - 2;
   cout << "x = " << x << endl;
   cout << "y = " << y << endl;
   cout << "rx = " << rx << endl;
}
```

- Изучите ссылку rx, подумайте, что произошло при выполнении выражений

```
rx += y;
и
rx = rx - 2;
```

Задание 4. Передача параметров в функцию по ссылке

- Изучите представленную ниже программу обмена переменных, в которой при передаче параметров реализуется передача их ссылок, и поэтому функция может изменять передаваемые переменные.

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

void swap(int &, int &); // прототип показывает, что функция
                         //принимает две ссылки на целый тип

int main()
{
  int x = 5, y = 10;
  cout << "main, Before swap, x: " << x << " y: " << y << endl;
  swap(x,y);  // вызов функции с передачей параметров
  cout << "main. After swap, x: " << x << " y: " << y << endl;
return 0;
}
```

```
void swap (int &x, int &y) // реализация функции
{
  int temp;
  cout << "Swap. Before swap, x:" << x << " y: " << y << endl;
  temp = x;
  x = y;
  y = temp;
  cout << "Swap. After swap, x: " << x << " y: " << y << endl;
}
```

Таким образом, благодаря использованию ссылок функция приобретает возможность изменять исходные данные в вызывающей функции, хотя при этом сам вызов функции ничем не отличается от обычного.