# САОД. Задание 3

## Темы

* классы и объекты,
* Inline методы,
* конструкторы “по умолчанию” и “инициализации”,
* математическая библиотека cmath.

## Задания

Если по расписанию еще не было лекции 2, то начните с ее беглого просмотра (15-20 мин) до слайда 16.

1. Запустить Visual Studio и создать консольный проект “classes”.
   1. Используя мастер добавления классов создать класс Complex.   
      *На имени проекта в контекстном меню “Добавить” -> “Класс”.  
      Определить имя класса Complex и завершить создание.*
   2. Изучить, что сделал мастер.
      1. Добавлены 2 файла Complex.cpp и Complex.h. Они и образуют модуль класса Complex.
      2. В заголовочном файле определение класса, в cpp- реализация (конструктора и деструктора);
      3. Обратите внимание как конструктор и деструктор реализованы в cpp. используется контекст класса Complex:: Complex () и Complex::~ Complex ()  
         *Это необходимо. Иначе компилятор не поймет, что функции относятся к классу, а не определены “сами по себе”*.

*Часто разработчики вообще не пользуются мастером создания классов, а добавляют необходимые файлы и определения “вручную”.*

1. Разработать “полезный” класс комплексных чисел.  
   Вспомните, что значит “***inline***” и как определяются *inline*-члены класса. В нашем классе все методы будут короткими и для эффективности имеет смысл сделать их именно *inline*.  
   Поэтому файл *Complex.cpp* нам не понадобится.
   1. Правой кнопкой мыши на имени файла (*Complex.cpp* в Обозреватель решения) и “*Удалить”*.
      1. В Complex.h удалить деструктор. **Для данного класса он не нужен. (почему?)**.
      2. Прототип конструктора заменить его реализацией, просто убрав ‘;’ и добавив {}
   2. Добавить мнимую и действительную часть в наш класс. В Complex.h   
      - перед конструктором добавить 2 публичных поля: double Re, Im;   
      - в конструкторе задать начальные значения полей Re и Im равными 0;  
      Проверить, что проект собирается, если надо устранить ошибки.
   3. В основной модуль (с main) добавить #include “complex.h” и создать объект типа Complex c; в main.
   4. Проверить, что проект собирается, если надо устранить ошибки.

*Пока класс почти бесполезен. Может только хранить действительную и мнимую части.*

1. Научить Complex выводить объекты в поток вывода. Для этого:
   1. Добавить в заголовочный файл Complex.h ниже объявления класса переопределение операции << для потока вывода и комплексного числа  
      inline ostream& operator << (ostream &o, const Complex &c)   
      { return o << '(' << c.Re << ", " << c.Im << ')'; }

*Теперь для комплексных чисел можно cout << c, т.к. cout является производным от ostream.*

* 1. Разберитесь с переопределением оператора.   
     *Обратите внимание на то, что вне класса, чтобы объявить inline-функцию перед ней обязательно указать inline!*  
     **Зачем возвращается ссылка на поток?   
     Почему параметр ‘c’ передается как константная ссылка?**

В основном модуле проверить в main, что вывод работает.

*С нашим классом работать пока не удобно. Нужно объявить (создать) объект, присвоить значения его полям.*

1. Доработать конструктор так, чтобы он мог создавать объекты сразу с нужными значениями (конструктор инициализации).
   1. В определении конструктора добавить 2 параметра double re = 0 и double im = 0, со значениями по умолчанию.   
      В теле конструктора использовать их как начальные значения (вместо 0).  
      *Значения по умолчанию позволят нам вызвать конструктор и без параметров, и с одним параметром.*

Проверьте, что в main теперь можно создавать объекты с нужными начальными значениями.

Complex a(1,2), b = 3;

cout<<a<<", "<<b<<endl;

1. Определить основные операции для Complex.
   1. Добавить операцию +.  
      *В определении класса ниже конструктора пишем*:  
      Complex operator +(const Complex&c)const   
      { return Complex (Re + c.Re, Im + c.Im); }

**Что означают квалификаторы const в определении операции?**

Теперь наши числа можно складывать.

* 1. Самостоятельно добавьте операции -, \*, /.   
     *Для умножения (x1+i\*y1)\*(x2+i\*y2) = (x1\*x2 – y1\*y2, i\*(x1\*y2+x2\*y1))  
     Для деления (x1+i\*y1)/(x2+i\*y2) =   
     ((x1\*x2 + y1\*y2)/ (x2\*x2 + y2\*y2), i\*(-x1\*y2+x2\*y1)/ (x2\*x2 + y2\*y2)).*

Проверьте, что операции +- и \*/ взаимно обратны!

(*если число вычесть из самого себя, то получится 0, если разделить на само себя, то получится 1*)

* 1. Самостоятельно реализуйте метод сопряжения комплексного числа   
     Complex Conjugate() – возвращает сопряженное число. (a + *i*\*b) => (a - *i*\*b)

1. Математические функции собраны в библиотеке cmath.
   1. Включить модуль математических функций <cmath>. *Лучше это делать сразу после #include <iostream>, т.е. перед using namespace std; Это сделает текст более наглядным.*
   2. Реализовать самостоятельно метод double Mod(), который возвращает модуль комплексного числа, sqrt(Re2 + Im2).
   3. Самостоятельно реализовать метод Arg(), который возвращает аргумент комплексного числа в экспоненциальном представлении. *a + i\*b = m\*exp(i\*arg)*. *Можно восстанавливать аргумент, например, так:  
      - если модуль равен 0, то возвращаем 0;  
      - если Re>0, то atan(Im/Re);  
      - если Re < 0, то для Im >= 0 acos(-1)+ atan(Im/Re)  
      - для Im < 0 -acos(-1)- atan(Im/Re)  
      - иначе acos(0 ) для Im > 0  
       - acos(0) для Im < 0*

*Получили довольно неплохой класс для работы с комплексными числами. Практически, ничем не отличается от встроенных int, double и пр.!*

1. Объявить массив комплексных чисел. *Здесь нам и понадобится конструктор по умолчанию, который можно вызывать без параметров.*
   1. Вывести массив в одну строку на консоль.
   2. Добавить инициализатор массива и повторить вывод.  
      Complex v[4]{ 1,2, Complex (2,3) };

Все как со встроенными типами!

1. Работа с динамическими блоками памяти.
   1. Объявить указатель на Complex \*pc.
   2. Создать объект Complex оператором new. Поиграть с параметрами.  
      new Complex(1), Complex, Complex(1,2)
   3. Вывести число на консоль.
   4. Вывести отдельно мнимую и действительную часть используя оператор ->
   5. Не забывайте delete после использования;
   6. Создать массив комплексных чисел (new Complex[3]);
   7. Разобраться, что означает pc[1].Re.
   8. Не забывайте delete после использования;

Завершайте работу, удалите ненужные файлы, архивируйте решение.

Теперь мы умеем:

* Добавлять и использовать классы;
* Делать их методы inline или обычными;
* Переопределять операции;
* Использовать cmath.

# Отчет

Подготовьте в Word краткий (*не более 1 стр.*) отчет о выполнении задания.

В отчете: дайте ответы на вопросы пп. 2 и 3 Задания, объясните, что такое inline-функции и зачем они нужны, как создаются динамические объекты и массивы объектов в C++, как они уничтожаются.

Отчет и архив решения поместите как ответ на задание.