Объектно-ориентированное программирование на c++
Лекция 1

Автор курса:

доцент Разумовский Алексей Игоревич

oopCpp@yandex.ru

Книги и ссылки

1. Страуструп Б. Язык программирования C++. Москва: Вильямс, 2011
2. Мейерс С. - Эффективный и современный C++. 42 рекомендации по использованию C++11 и C++14, 2016
3. Д. Элджер - Библиотека программиста C++ - Гарвард: World Group, 2004.
4. **Стив Макконнелл Совершенный код**
5. **http://www.cplusplus.com**

Методология познания

Метод проб и ошибок.

Чем больше сделаете ошибок, тем быстрее научитесь.

Интегрированная Среда Разработки (IDE) для C++:

Microsoft Visual Studio vv. 2019-2022

Исследования, проведенные в 1970-х годах в Массачусетском технологическом институте и исследовательском центре Xerox Palo Alto Research Center, привели к разработке философии объектно-ориентированного программирования и созданию языков реализации, включая Smalltalk, Java, C++.

Предполагается, что вы уже знаете понятия цикла (**for, while, do{ }while**), главной функции (**main/WinMain**), операторов **if/else, switch, return.**

Также предполагается, что вы знаете, что такое указатель, ссылка и их взаимоотношения, а также понимаете арифметику указателей, включая операторы инкремента (++) и декремента (--).

Я буду постепенно добавлять новые термины и использовать их в примерах.

Понятия типа, класса, полиморфной иерархии, глубокого копирования, контейнеров, итераторов, адаптеров, функторов, умных указателей, потоков выполнения будут раскрываться последовательно от лекции к лекции.

Запреты использования.

При решении задач на лабах / семинарах и в курсовой работе запрещено применять ключевое слово **auto**, а также двухсекционный цикл for : **for (auto l : v)** .

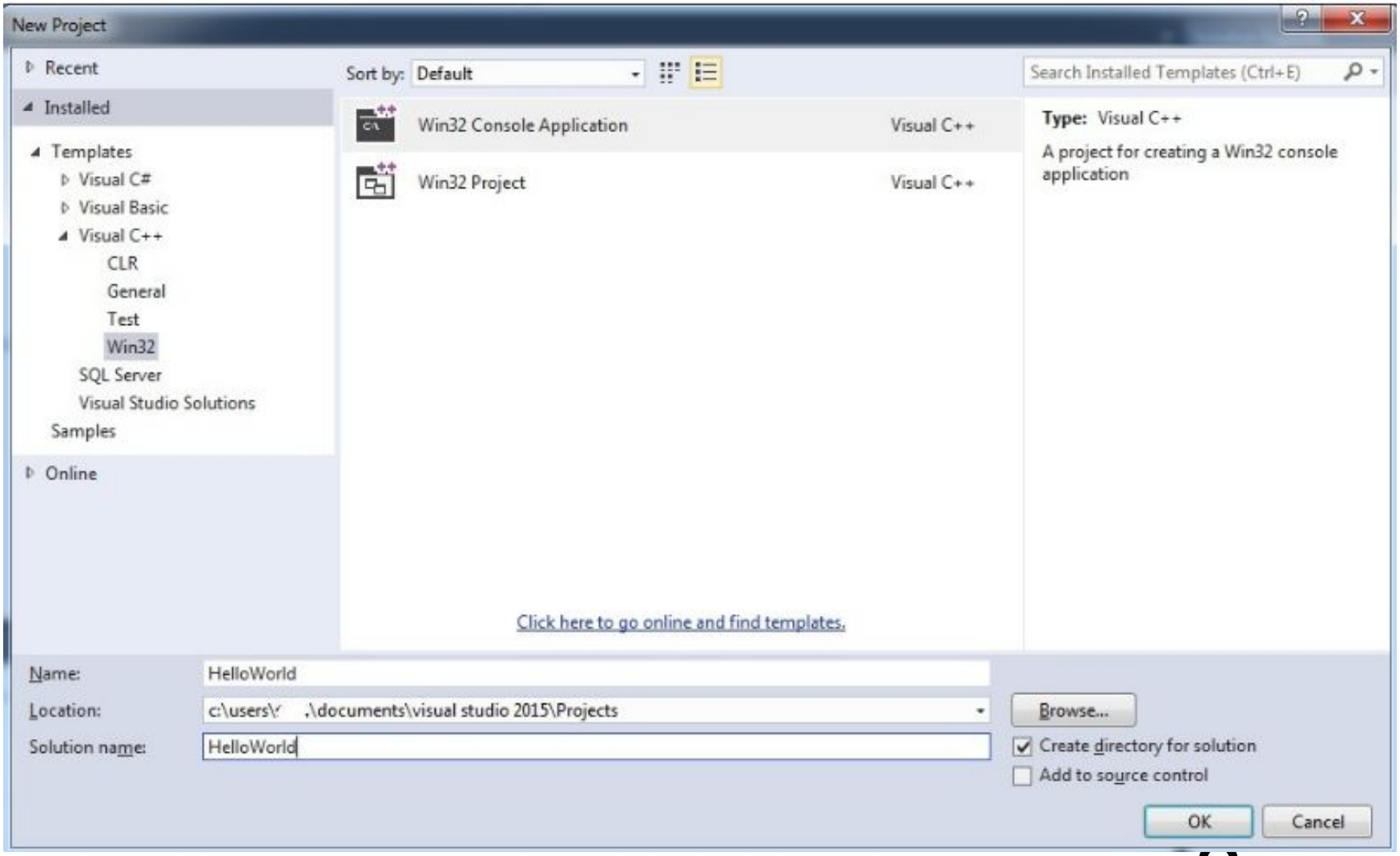
for следует пользоваться только трехсекционным, например:

for (T t = begin() ; t != end(); ++t)

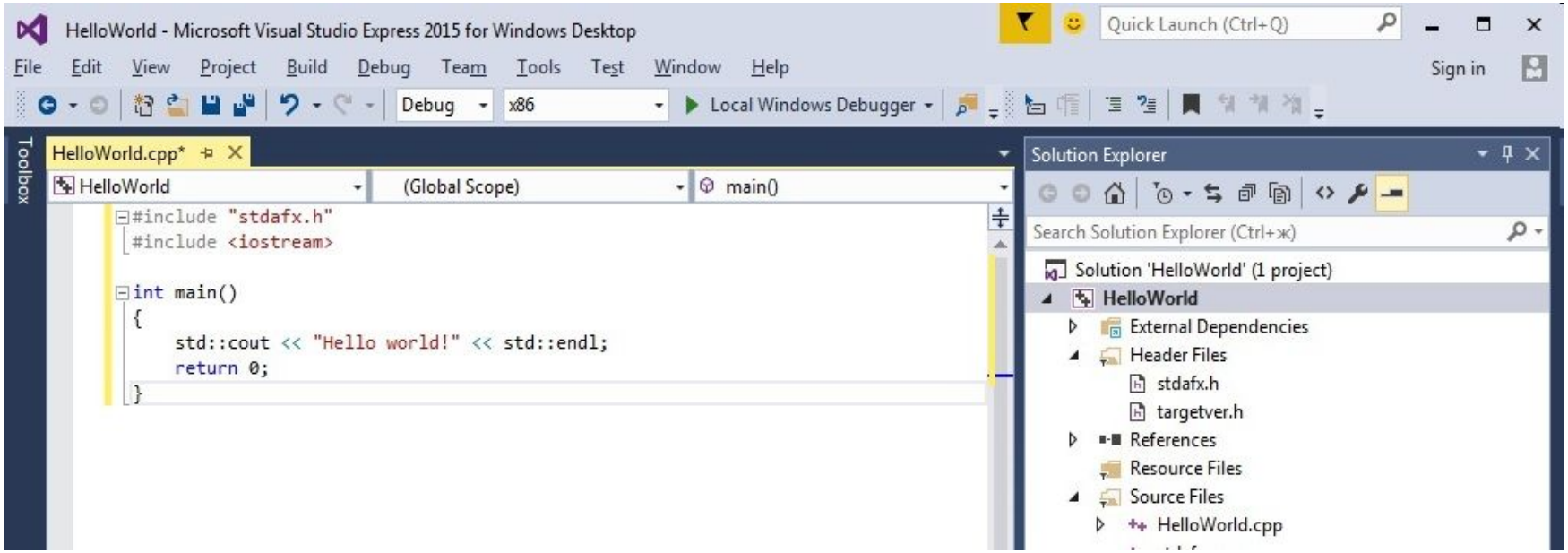
Чего нет в книгах и учебниках

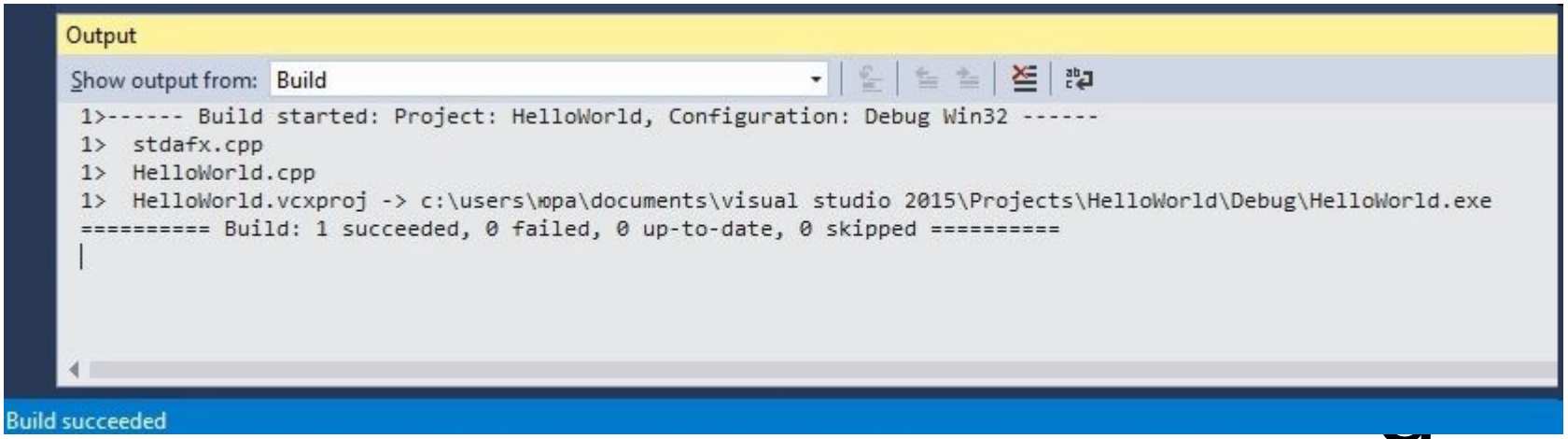
- Каждый вызов функции – это отражение вашего желания получить нужный результат. Это здравый смысл операции!
- Каждый элемент программы несет в себе ответственность за его использование. Поэтому, например, применение умных указателей, которые сами могут освободить выделенную память – лишает программиста ответственного поведения.

Создание проекта



Компиляция проекта





В начале файла программы следует указать нужные хидеры:

- #include<iostream>
- #include<fstream>
- #include<string>
- #include<vector> // или set или map
- #include<algorithm>
- #include<cmath>
- **using namespace std;** // использовать пространство имен std

Файлы

3 способа доступа к файлам данных:

- Windows API: CreateFile (и другие функции для работы с файлами)
- C: fopen (и прочие функции для работы с файлами из stdio.h)
- C++: fstream (ifstream, ofstream) (из stl)

Потоки данных

- C: printf и scanf (#include<stdio.h>)
- C++: cin и cout (#include<iostream>)

Файловые потоки:

(#include<fstream>)

Пример

```
#include <fstream>
#include <libc.h>

void error (char* s, char* s2 ="")
{
   cerr << s << ' ' << s2 << '\n';
   exit(1);
}

int main(int argc, char* argv[])
{
    ifstream from(“file1.txt”);
    ofstream to(“file2.txt”);
              char ch;
    while (from.get(ch)) to.put(ch);

    if (!from.eof() || to.bad())
       error("something strange happened");

    return 0;
 }
```

Стандартная библиотека шаблонов STL

Включает, основанную на методологии обобщенного программирования библиотеку классов, содержащую:

- Контейнеры (для хранения данных произвольного типа)
- Итераторы (для осуществления доступа к данным контейнеров)
- Алгоритмы
- Адаптеры контейнеров, итераторов и функционалов.
- Стримы (streams) для выполнения потоковых операций с данными
- Потоки исполнения (threads) и элементы синхронизации в потоках
- Прочие элементы, например, умные указатели, лямбда, диапазоны

Пример stl-алгоритма: сортировка

```
#include<vector>
#include<algorithm>
using namespace std;

vector <int> v;
for ( int i=10; i< 0; i--)
   v.push_back(i);                // заполняем вектор: 10,9,8,7,6,5,4,3,2,1

vector <int>::iterator it= v.begin();

sort (v.begin(),v.end());         // сортируем данные вектора 1,2,3,4,5,6,7,8,9,10
```

Три основных свойства ООП

- Абстракция (данных) (отвлечение)
- Инкапсуляция (скрытие)
- Полиморфизм (разнообразие)

(но НЕ наследование и НЕ модульность)

В следующих лекциях - подробности

Пример ООП- программы

```
  class A{
public:
A(){ }
virtual ~A(){ }
 virtual void print(){cout<<"a"<<endl;}
};

class B: public A{
public:
  void print(){cout<<"b"<<endl;}я
};
 void print_all ( A** v  , int size )  {
      // распечатать все
          int i = 0;
            while ( i<size )
            {
               (v [ i ]) -> print (    );
              i = i + 1;
            }
 }
```

```
int _tmain(int argc, _TCHAR* argv[]){
 A a;
B b;
 A m[2];
 m[1]= b;
A*m0[2]={&m[0],&m[1]};
     print_all(m0,2);
      A*m1[2];
  m1[0]=new A;
  m1[1]= new B;
           print_all( m1,2);
return 1;
}
```

// Результат выполнения программы:

a

a

a

b

Контейнеры

Контейнеры - это объекты, которые **содержат** другие **объекты**. Они управляют размещением в памяти и свобождением этих объектов через конструкторы, деструкторы, операции вставки и удаления.

Есть два основных вида контейнеров: последовательности и ассоциативные контейнеры.

Последовательности (Sequences)

- Вектор (vector)
- Список (list)
- Двусторонняя очередь (deque)

Ассоциативные контейнеры (Associative containers)

- Множество (set)
- Множество с дубликатами (multiset)
- Словарь (map)
- Словарь с дубликатами (multimap)

vector<T>

Векторы -

представляют собой контейнеры последовательностей, представляющие массивы, которые могут меняться по размеру.

Подобно массивам, векторы используют **смежные места хранения** для своих элементов, что означает, что их элементы также могут быть доступны с помощью смещений на обычных указателях к его элементам и так же эффективно, как и в массивах. Но в отличие от массивов их размер может изменяться динамически, при этом хранилище автоматически обрабатывается контейнером.

Внутри векторы используют динамически выделенный массив для хранения своих элементов. Этот массив, возможно, потребуется перераспределить для увеличения размера при вставке новых элементов, что подразумевает выделение нового массива и перемещение в него всех элементов. Это относительно дорогостоящая задача с точки зрения времени обработки, и, следовательно, векторы не перераспределяются каждый раз, когда элемент добавляется в контейнер.

Пример

```
#include <iostream>
#include <vector>
int main (){  // constructors used in the same order as described above:
  std::vector<int> first;                          // empty vector of ints
  std::vector<int> second (4,100);                 // four ints with value 100
  std::vector<int> third (second.begin(),second.end());  // iterating through second
  std::vector<int> fourth (third);                 // a copy of third
          // the iterator constructor can also be used to construct from arrays:
  int myints[] = {16,2,77,29};
  std::vector<int> fifth (myints, myints + sizeof(myints) / sizeof(int) );
  std::cout << "The contents of fifth are:";
  for (std::vector<int>::iterator it = fifth.begin(); it != fifth.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';
  return 0;
}
```

Output:
The contents of fifth are: 16 2 77 29

push_back

```
#include <iostream>
#include <vector>
int main () {
 std::vector<int> myvector;
  int sum (0);
  myvector.push_back (100);
  myvector.push_back (200);
  myvector.push_back (300);

  while ( ! myvector.empty())
  {
    sum+=myvector.back();
    myvector.pop_back();
  }

  std::cout << "The elements of myvector add up to " << sum << '\n';
  return 0;
}
```

Output: The elements of myvector add up to 600

vector::operator [ ] и vector::at

Возвращает ссылку на элемент в позиции n в векторном контейнере.

Аналогичная функция-член, vector :: at, имеет то же поведение, что и эта операторная функция, за исключением того, что vector :: at проверен на границах и сигнализирует, если запрошенная позиция вне допустимого диапазона, выбрасывая исключение out_of_range.

reference operator [ ] (size_type n);

const_reference operator [ ] (size_type n) const;

reference at (size_type n);

const_reference at (size_type n) const;

Пример

```
#include <iostream>
#include <vector>
int main () {
      std::vector<int> myvector  (10);   // 10 zero-initialized elements
      std::vector<int>::size_type  sz = myvector.size();
 for (unsigned i=0; i<sz; i++) myvector[i]=i; // assign some values:
     // reverse vector using operator [ ]:
  for (unsigned i=0; i<sz/2; i++)   {
     int temp;
    temp = myvector[sz-1-i];
    myvector[sz-1-i] = myvector[i];
    myvector [i] = temp;
  }
      std::cout << "myvector contains:"<< '\n';
   for (unsigned i=0; i<sz; i++)
      std::cout << ' ' << myvector[i];

std::cout << '\n';
  return 0; }
```

Output:
myvector contains:
 9 8 7 6 5 4 3 2 1 0

Отладчик MS Visual Studio

Клавиши отладки (основные):

F9 – поставить или снять точку останова программы

F10 – совершить одно отладочное действие: выполнить одну результирующую операцию.

F11 – войти внутрь функции

SHIFT + F11 – выйти из функции

Окна отладки:

В процессе отладки программы можно открывать большинство окон отладчика. Чтобы просмотреть список окон отладчика, установите точку останова и начните отладку. Когда точка останова будет достигнута и выполнение остановится, выберите пункт Отладка / Окна.

Окно	Сочетание клавиш	Раздел
Breakpoints	CTRL+ALT+B	Use Breakpoints
Exception Settings	CTRL+ALT+E	Manage Exceptions with the Debugger
Output	CTRL+ALT+O	Output Window
Watch	CTRL+ALT+W, (1, 2, 3, 4)	Watch and QuickWatch Windows
QuickWatch	SHIFT+F9	Watch and QuickWatch Windows
Autos	CTRL+ALT+V, A	Autos and Locals Windows
Locals	CTRL+ALT+V, L	Autos and Locals Windows
Call Stacks	CTRL+ALT+C	How to: Use the Call Stack Window
Immediate	CTRL+ALT+I	Immediate Window
Threads	CTRL+ALT+H	Debug using the Threads Window
Modules	CTRL+ALT+U	How to: Use the Modules Window
Tasks	CTR:+SHIFT+D, K	Using the Tasks Window
Processes	CTRL+ALT+Z	Debug Threads and Processes
Memory	CTRL+ALT+M, (1, 2, 3, 4)	Memory Windows
Disassembly	CTRL+ALT+D	How to: Use the Disassembly Window
Registers	CTRL+ALT+G	How to: Use the Registers Window

Окно	Сочетание клавиш	Раздел
Точки останова	CTRL+ALT+B	Использование точек останова
Параметры исключений	CTRL+ALT+E	Управление исключениями с помощью отладчика
Вывод	CTRL+ALT+O	Окно выходных данных
Контрольное значение	CTRL+ALT+W, (1, 2, 3, 4)	Окна "Контрольные значения" и "Быстрая проверка"
Контрольное значение	SHIFT+F9	Окна "Контрольные значения" и "Быстрая проверка"
Авто	CTRL+ALT+V, A	Окна переменных
Локальные	CTRL+ALT+V, L	Окна переменных
Интерпретация	CTRL+ALT+I	Окно интерпретации
Стеки вызовов	CTRL+ALT+C	Использование окна стека вызова
Потоки	CTRL+ALT+H	Использование окна потоков
Модули	CTRL+ALT+U	Использование окна модулей
Задачи	CTRL+SHIFT+D, K	Использование окна задач
Процессы	CTRL+ALT+Z	Отладка потоков и процессов
Память	CTRL+ALT+M, (1, 2, 3, 4)	Окно памяти
Дизассемблированный код	CTRL+ALT+D	Использование окна дизассемблирования
Регистры	CTRL+ALT+G	Использование окна регистров

Для желающих быстро научиться - проекты

На следующих слайдах – 15 проектов, простых! Кто желает, можете их сделать и прислать мне на почту до следующей лекции (крайний срок — вторник 12.09, если после 24-00, то не приму).

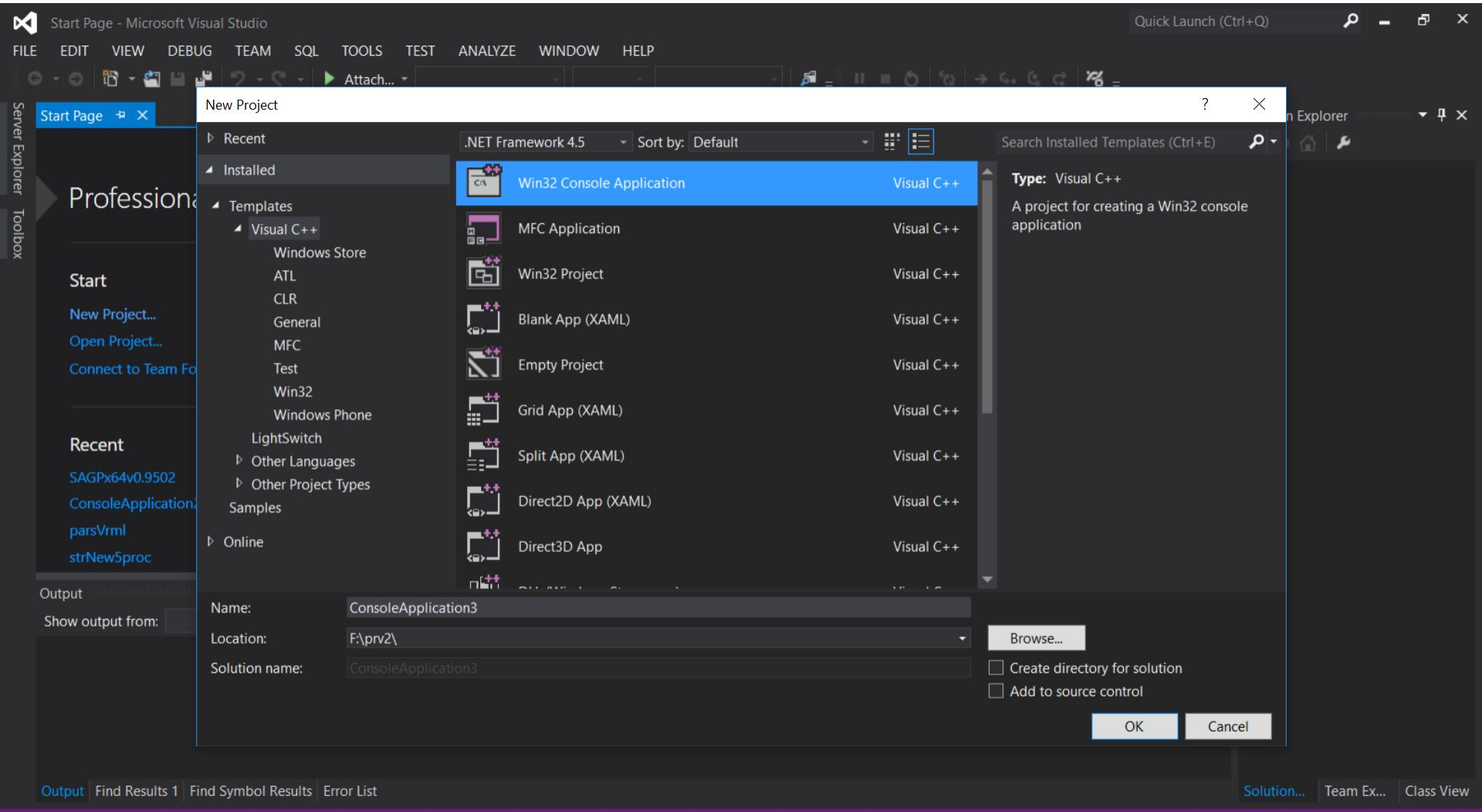
На каждой лекции я буду давать по домашнему заданию.

Кто станет выполнять задания и присылать их результаты регулярно мне – получит на экзамене автоматом отличную оценку.

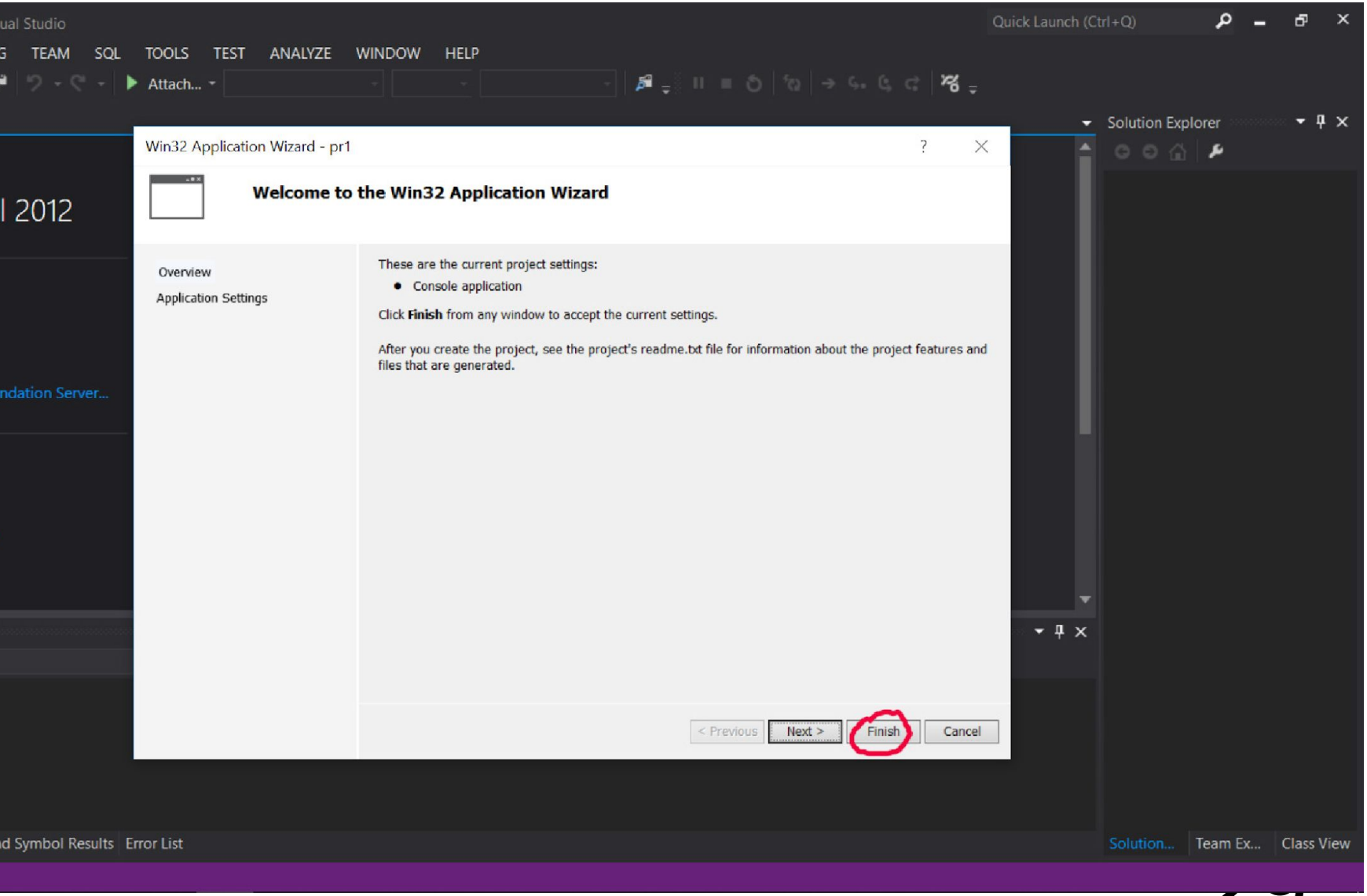
Кроме того, можете приходить ко мне заниматься индивидуально, только предварительно надо договориться – шлите письмо.

Также можете присылать любые вопросы в любое время.

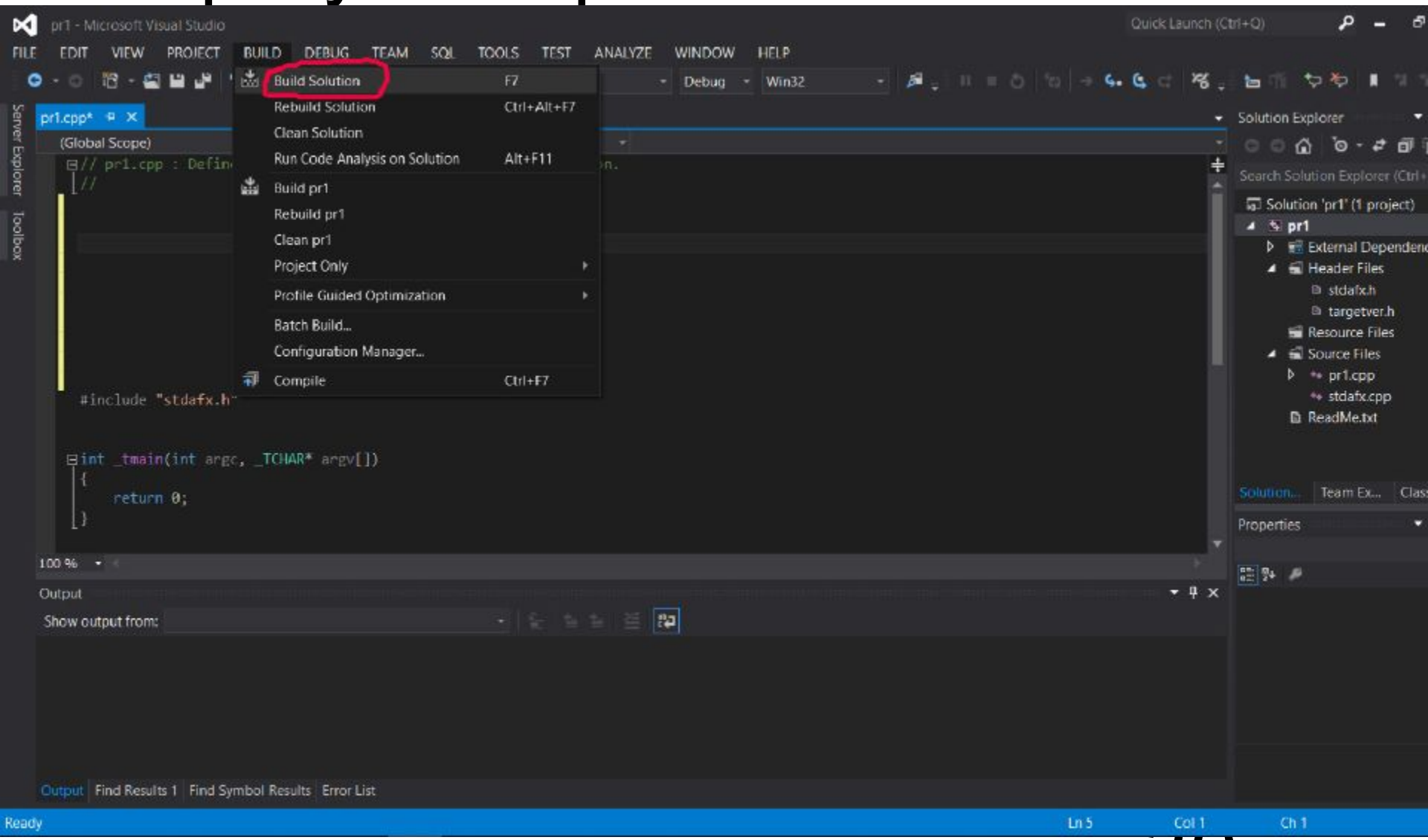
Домашнее задание. Создаем проект 1



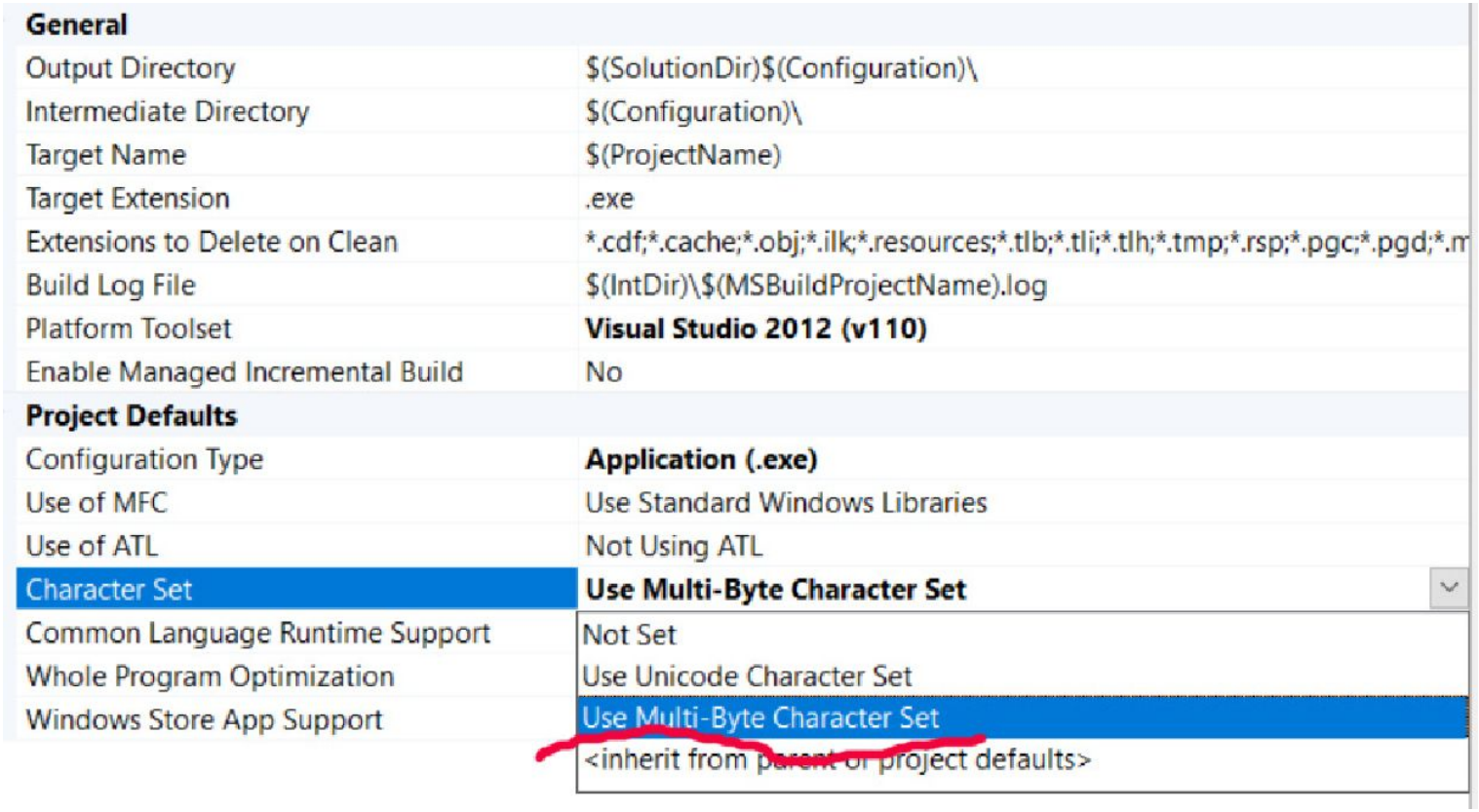
Сразу как задали имя проекта выбираем кнопку Finish



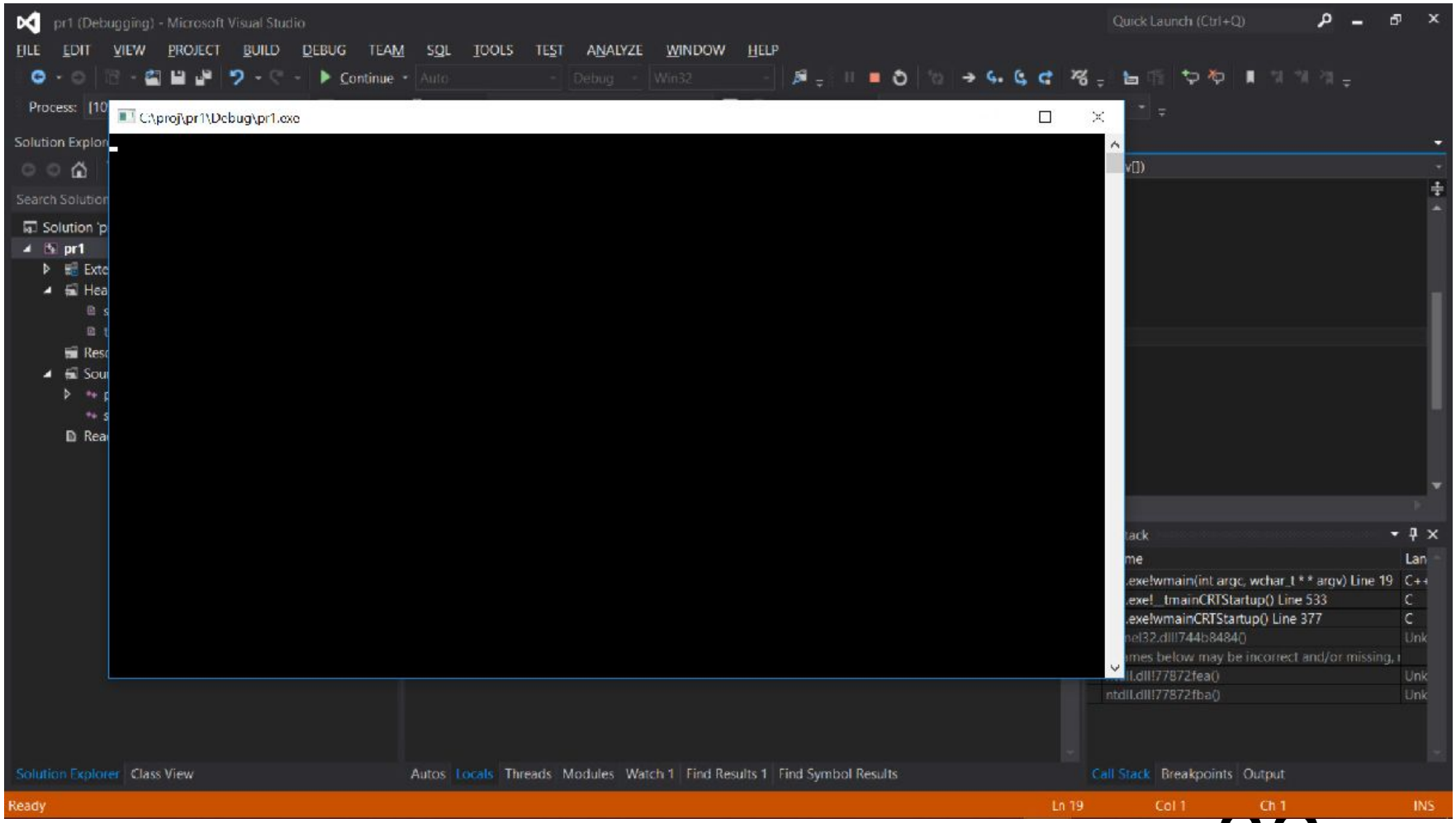
Ничего не делаем, сразу выбираем Build Solution



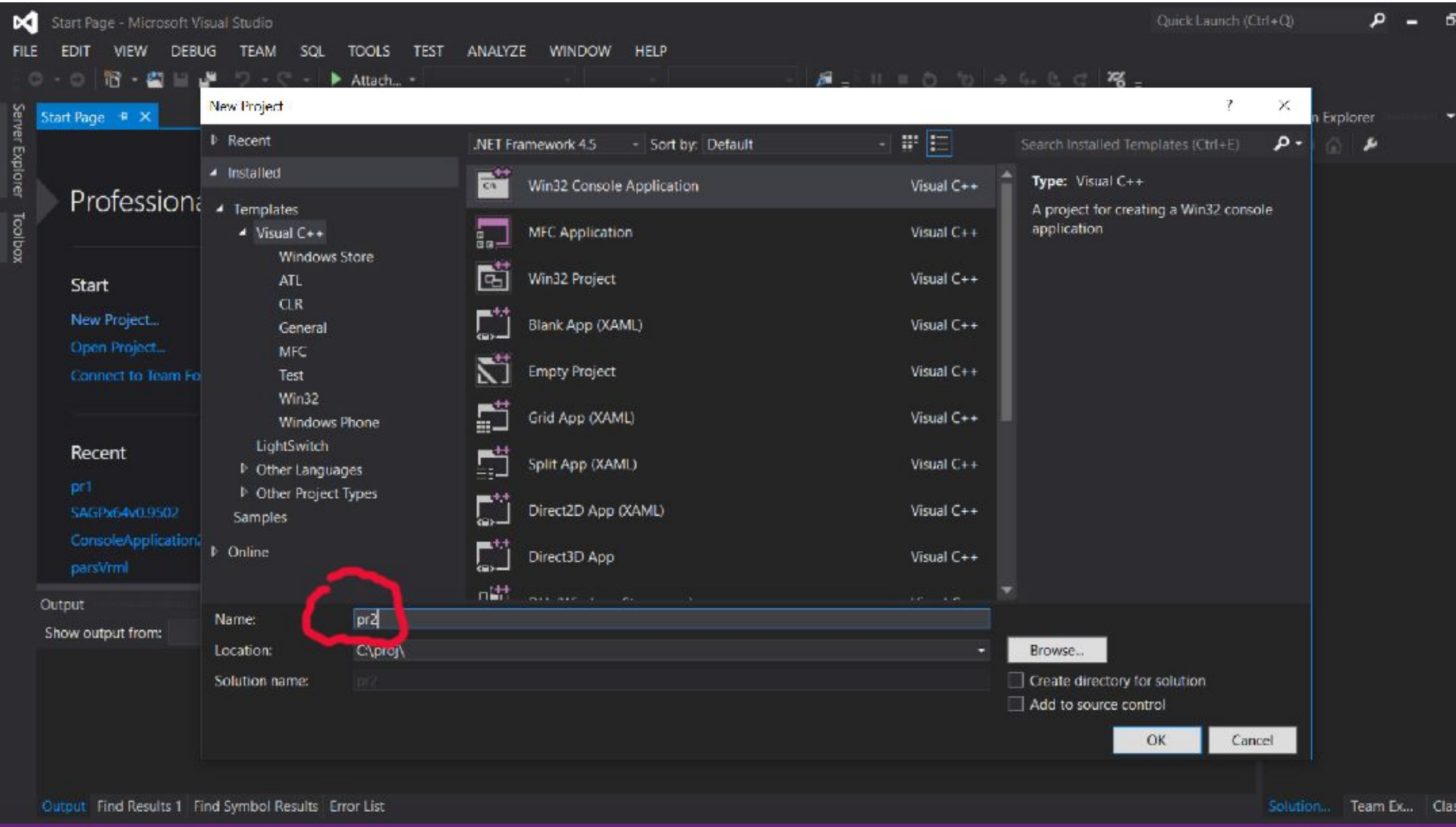
В свойствах проекта меняем Unicode на Multi Byte



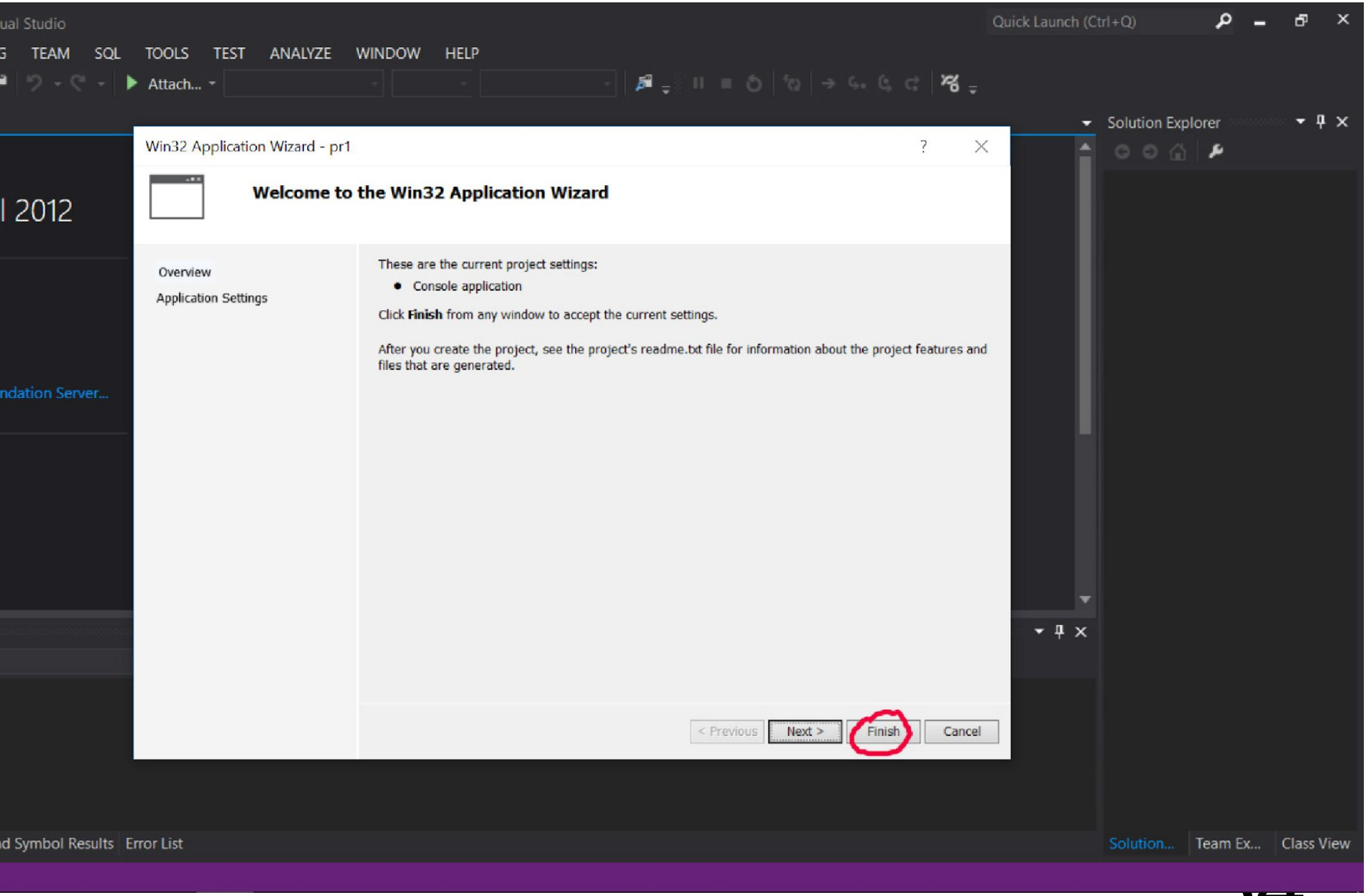
Запускаем исполнение программы (клавиша F5 или через меню) – если ошибок при создании программы нет, то:



Создаем проект 2



Сразу как задали имя проекта выбираем кнопку Finish



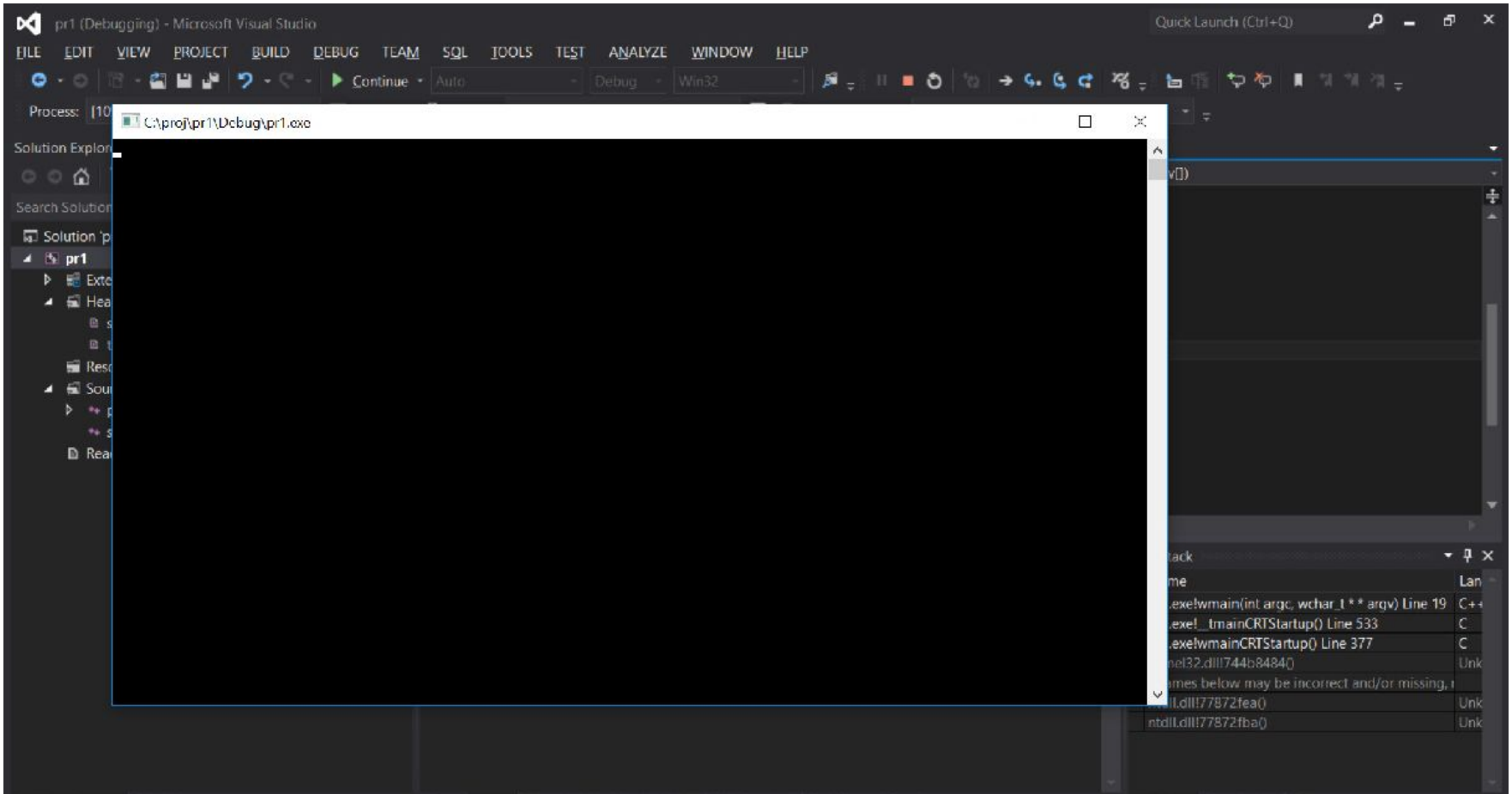
Добавляем заголовочные файлы, получаем код:

```
#include "stdafx.h"
#include<iostream>
#include<fstream>
#include<string>
#include<set>
#include<map>
#include<vector>
#include<algorithm>
#include<cmath>
using namespace std;


int _tmain(int argc, _TCHAR* argv[])
{

return 0;
}
```

Запускаем исполнение программы (клавиша F5 или через меню) – если ошибок при создании программы нет, то получаем пустое черное окно, которое быстро закроется (или не закроется)



Создаем проект 3 – pr3, в котором добавляем заголовочные файлы, как в проекте 2. Кроме того, добавляем строки в главную функцию программы:

```
int _tmain(int argc, _TCHAR* argv[])
{
string s;
s= "test";
cout<<s<<endl;
 getchar();
return 0;
}
```

Затем запускаем построение программы и затем саму программу

Опять, если нет ошибок, должно открыться черное окно, которое теперь не закроется, а выведет строку **test**.
Закрыть окно можно нажав клавишу «Return»

Создаем проекты 4, 5 и 6, в которые добавляем по одному новому элементу. Смотрим результат.

Новые элементы соответственно:

для 4-го

```
 ofstream of;  // создает объект-поток
of.open("myfile.txt");// создает файл
for(int i=0;i<10;i++){
 of<<i<<endl; // записывает в файл
}
// откройте файл и посмотрите внутрь
```

для 5

```
ifstream ifile;  // создает объект-поток
ifile.open("myfile.txt");// открывает файл
string s0,s1;
   for(; ; ){
   ifile>>s0 ; // читает из файла
   if(  ifile.eof()) break;
   s1+=s0+"\n";
   }   cout<<s1;
```

для 6

```
ifstream ifile;  // создает объект-поток
 ifile.open("myfile.txt");// открывает файл
 int a;
vector <int> v;
for(; ; ){
ifile>>a ;
 if(  ifile.eof()) break;
v.push_back(a); // добавляем в вектор
}
for(int i=0; i<v.size(); i++){
 cout<<v[i]<<endl; // вывод данных
}
```

Создаем проект 7, в который также добавляем новый элемент, меняющий поведение программы

для 7-го: изучаем сортировку

```
 ifstream ifile;            // создает объект- файловый поток
ifile.open("myfile.txt"); // открывает файл
 int a;
vector <int> v;
for(; ; ){
   ifile>>a ;
 if(  ifile.eof()) break;
    v.push_back(a);
}

 sort(v.begin(),v.end());
 // sort(v.rbegin(),v.rend()); // раскомментируйте и сравните рез-т

for(int i=0; i<v.size(); i++){
 cout<<v[i]<<endl;
}
```

Создаем проект 8, в который также добавляем новый элемент

для 8 Обернем код классом:

```
class Reader{
vector <int> m_v;
public:
Reader(){ }
void read(const char* name);
void print();
};

void Reader::read(const char* name){
  ifstream ifile;
 ifile.open(name);
   int a;
 for(; ; ){
   ifile>>a ;
 if(  ifile.eof()) break;
    m_v.push_back(a);
 }
}

void Reader::print( ){
  for(int i=0; i<m_v.size(); i++){
    cout<<m_v[i]<<endl;
  }
}

int _tmain(int argc, _TCHAR* argv[])
{
Reader r;
r.read("myfile.txt");
r.print();

  getchar();

return 0;
}
```

Создаем проект 9, в который добавляем новый элемент – класс с деструктором

для 9:

```
class Test{
public:
 Test(){
  cout<<"Construct" <<endl;
 }
 ~Test(){
  cout<<"Destruct" <<endl;
 }
};
```

```
int _tmain(int argc, _TCHAR* argv[])
{
vector <Test*>v(10);
for(int i=0;i<v.size();i++){
v[i]=new Test();
}

for(int i=0;i<v.size();i++){
delete v[i] ; v[i]=0;
}
return 0;
}
```

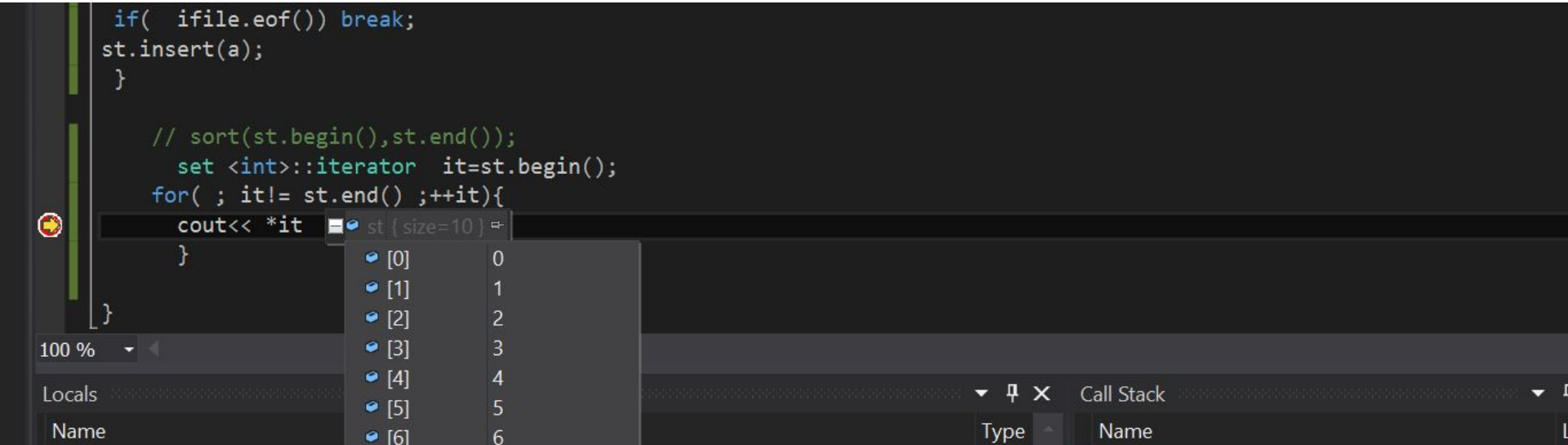
Можете попробовать закомментировать строку delete v[i] ; v[i]=0;
и увидите, что только на вас лежит забота о созданном объекте

Создаем проект 10, в котором также изучаем сортировку используя std::set

для 10

```
 void func(){
 ifstream ifile;
ifile.open("myfile.txt");
  int a;
  set <int> st;
for(; ; ){
  ifile>>a ;
 if(  ifile.eof()) break;
   st.insert(a);
 }
   set <int>::iterator  it=st.begin();
for( ; it != st.end() ;++it){
   cout<< *it  <<endl;
  }
 }
int _tmain(int argc, _TCHAR* argv[])
{
  func();
  return 0;
}
```

Активно используем отладчик: клавиша F9 устанавливает точку остановки – в этот момент можно посмотреть, что внутри переменных

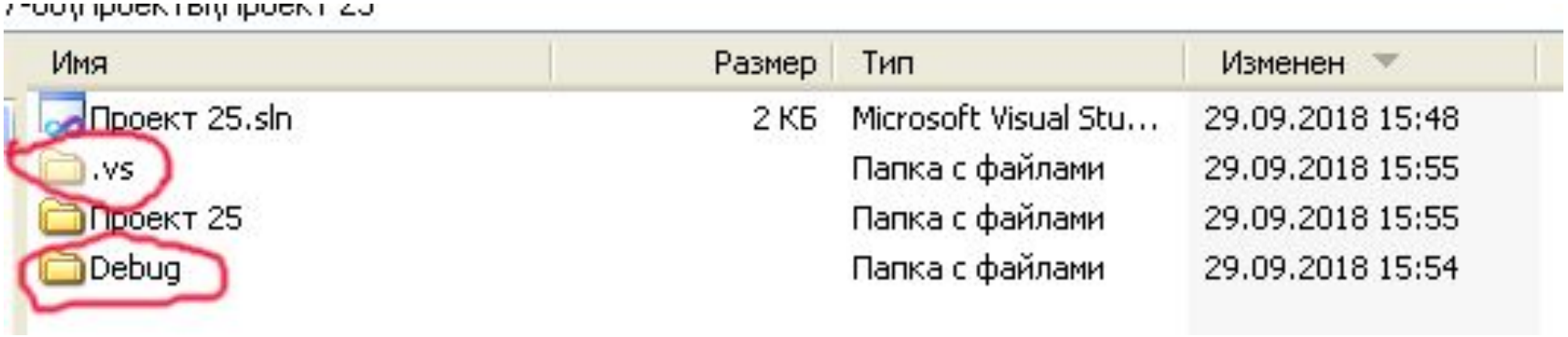


Создаем проекты с 10 по 15, **в которые также добавляем по одному элементу (любые на свой вкус), которые меняют поведение программы**

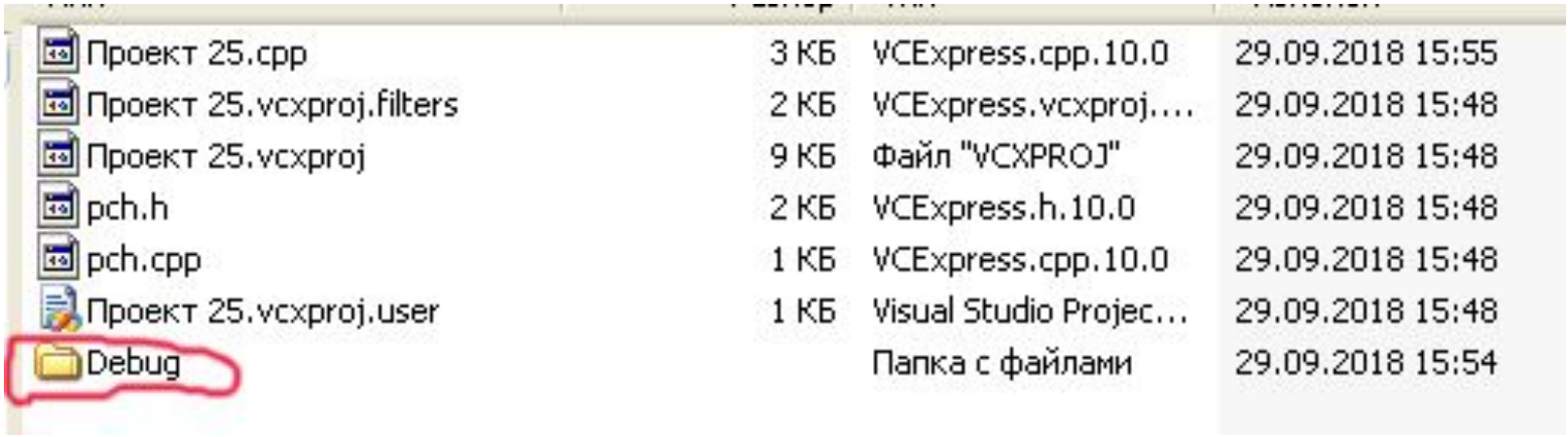
После создания всех проектов и проверки правильной работы программ, их нужно заархивировать, положить куда-нибудь в облако и прислать архив или ссылку на него мне на почту:

oopCpp@yandex.ru

В современных версиях следует удалить:



Имя	Размер	Тип	Изменен
Проект 25.sln	2 КБ	Microsoft Visual Stu...	29.09.2018 15:48
.vs		Папка с файлами	29.09.2018 15:55
Проект 25		Папка с файлами	29.09.2018 15:55
Debug		Папка с файлами	29.09.2018 15:54



Проект 25.cpp	3 КБ	VCExpress.cpp.10.0	29.09.2018 15:55
Проект 25.vcxproj.filters	2 КБ	VCExpress.vcxproj....	29.09.2018 15:48
Проект 25.vcxproj	9 КБ	Файл "VCXPROJ"	29.09.2018 15:48
pch.h	2 КБ	VCExpress.h.10.0	29.09.2018 15:48
pch.cpp	1 КБ	VCExpress.cpp.10.0	29.09.2018 15:48
Проект 25.vcxproj.user	1 КБ	Visual Studio Projec...	29.09.2018 15:48
Debug		Папка с файлами	29.09.2018 15:54

Пример задачи на диктанты, контрольные, лабораторные, семинары и коллоквиум

Создать полиморфную иерархию из двух классов — базового и производного. Внутри задать конструкторы с инициализацией доступной извне строки типа const char* именем класса, передавая его как аргумент конструктора. Написать деструкторы с выводом строки указывающей на принадлежность классу.

В главной функции программы создать несколько объектов указанных типов данных. Затем поместить их в хранилище типа vector<T>, где T — тип данных хранения полиморфных объектов.

Обратите внимание: чтобы единообразно хранить полиморфные объекты одной иерархии в общем хранилище, необходимо в качестве параметра шаблона задавать указатель на базовый класс.

vector < Base* >

а не

vector < Base > , как было бы в c#

Контрольная работа 1

Создать полиморфную иерархию из двух классов — **базового** и **производного** . Названия им дать по своим имени и фамилии латиницей. Внутри разместить по одному члену данных типа int и по конструктору, в котором присвоить значение этому члену данных – в каждом классе свое. Также определить в класс деструкторы. В полиморфной иерархии деструктор базового класса должен быть – виртуальным.

В главной функции программы создать по одному объекту ваших типов. Затем поместить их в хранилище типа vector<T>, где T — тип данных хранения полиморфных объектов, он должен быть указательным

```
vector < Base* > v.
```

Помещать их в вектор так:

```
Base* b= new  Base ;
v.push_back (b);
```

В конце главной функции – освободить память в цикле:

```
delete v[i];
```