Язык программирования Си

Бикулов Д.А., Иваницкая Н.В., Иванов А.П.

Семинар 1. Средства программирования. Структура консольного приложения, этапы компиляции и сборки программы. Первая программа.

1 Введение

- Общий обзор курса, порядок работы на семинаре и в практикуме, отчетность, техника безопасности, ответственность за сохранение индивидуального пароля в практикуме. Подчеркнуть обязательное наличие у студента учебника по языку программирования Си на каждом практическом занятии.
- Опрос, выясняющий текущий уровень знаний студентов группы по работе с компьютером и программированию.
- История создания языка Си. Рекомендуемая литература.

2 Интегрированная среда программирования

С первого же практического занятия студенты должны устойчиво освоить шаги, которые необходимо выполнить для заведения полноценного программного проекта в интерактивной среде программирования. Основные шаги, необходимые для этого представлены в настоящем разделе. Версии интерактивной среды программирования могут различаться, но разработчик обычно обеспечивает совместимость пользовательского интерфейса среды с предыдущими версиями.

Запуск среды программирования Microsoft Visual Studio выполняется через меню **«Пуск»** обычным образом, как правило, это не вызывает трудностей у студентов, нужно только показать, как найти среду программирования среди многообразия установленных программ.

Дальше студенты должны выполнить заведение проекта типа «консольное приложение» (Win32 Console Application, т.е. текстовое приложение), нужно пояснить, что без этих действий будет автоматически создан проект для графической программы Microsoft Windows, что выходит за рамки изучаемого материала. Ошибки заведения проекта являются одними из наиболее часто встречаемых у студентов, нужно проследить за тем, что все четко освоили эти действия.

Еще один существенный момент: нужно пояснить, что в дисплейном классе практикума от одного занятия до другого будут сохраняться только проекты, которые заводятся на диске Z: (домашний диск студента), проекты и любые другие данные и тексты программ пользователя, сохраненные в любых других местах – могут быть просто стерты до следующего занятия и их придется заводить и набирать заново.

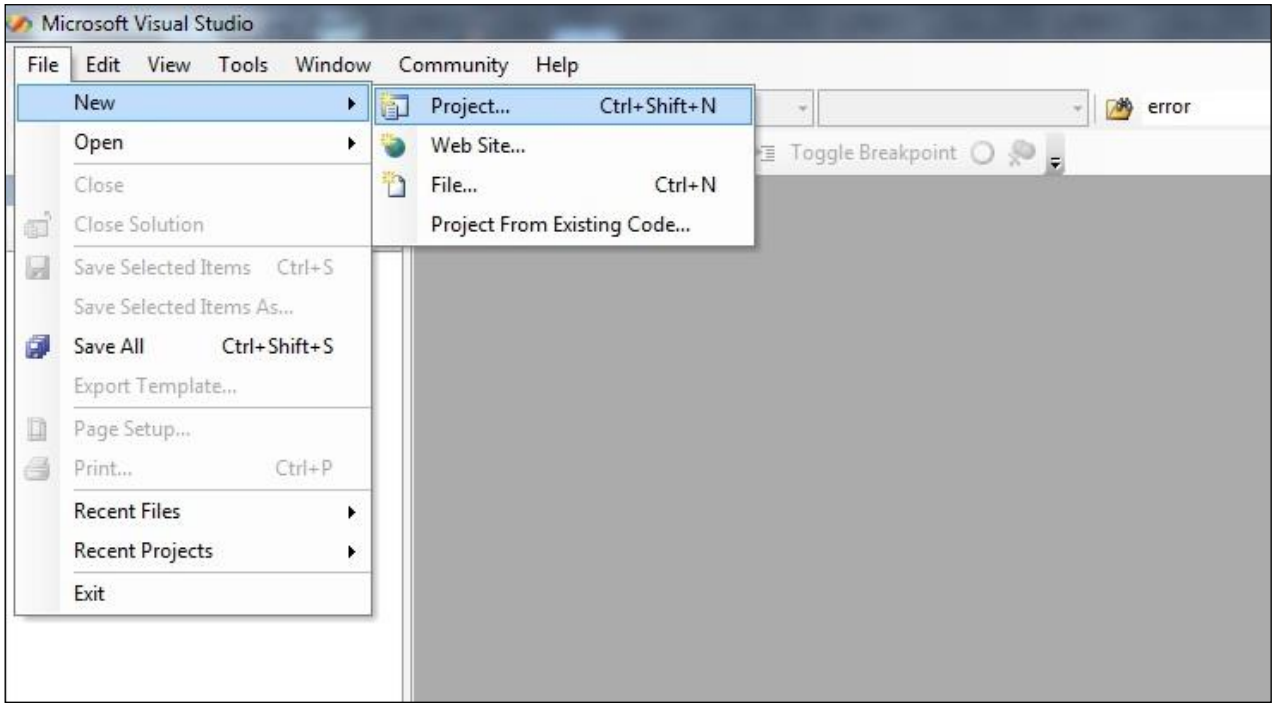
Студенты часто задают вопросы про английский язык среды программирования. Тут важно их успокоить: преподаватель в практикуме всегда поможет перевести непонятные сообщения интерактивной среды программирования, объяснит их смысл, но и сами студенты должны конспектировать и запоминать переводы наиболее частых сообщений об ошибках.

2.1 Пуск -> Все программы-> Microsoft Visual Studio 2008 -> Microsoft Visual Studio 2008

В появившемся окошке выбираем Visual C++ Development Settings, затем нажимаем кнопку Start Visual Studio. Этот запрос возникает не всегда, но если возникнет – нужно правильно на него ответить.

2.2 File -> New -> Project

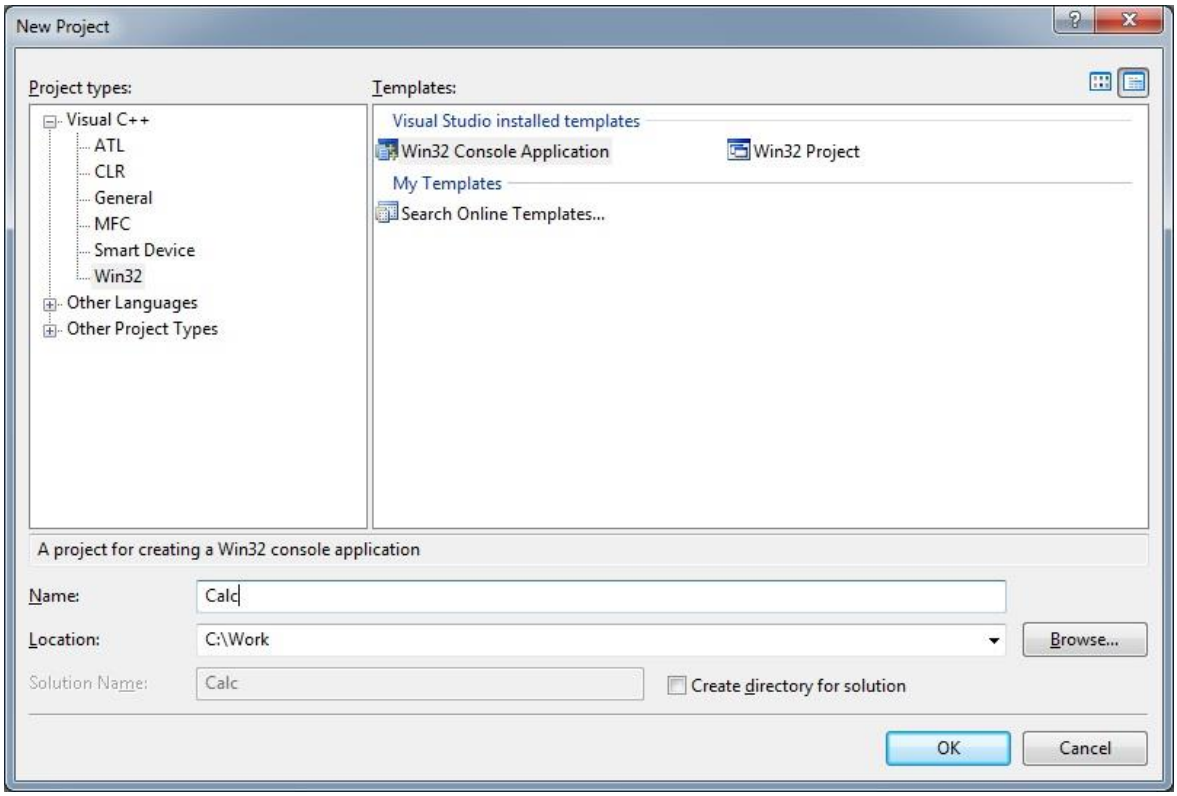
Данную последовательность действий нужно выполнить после запуска интерактивной среды программирования для заведения проекта.



2.3 win32 -> win32 console application

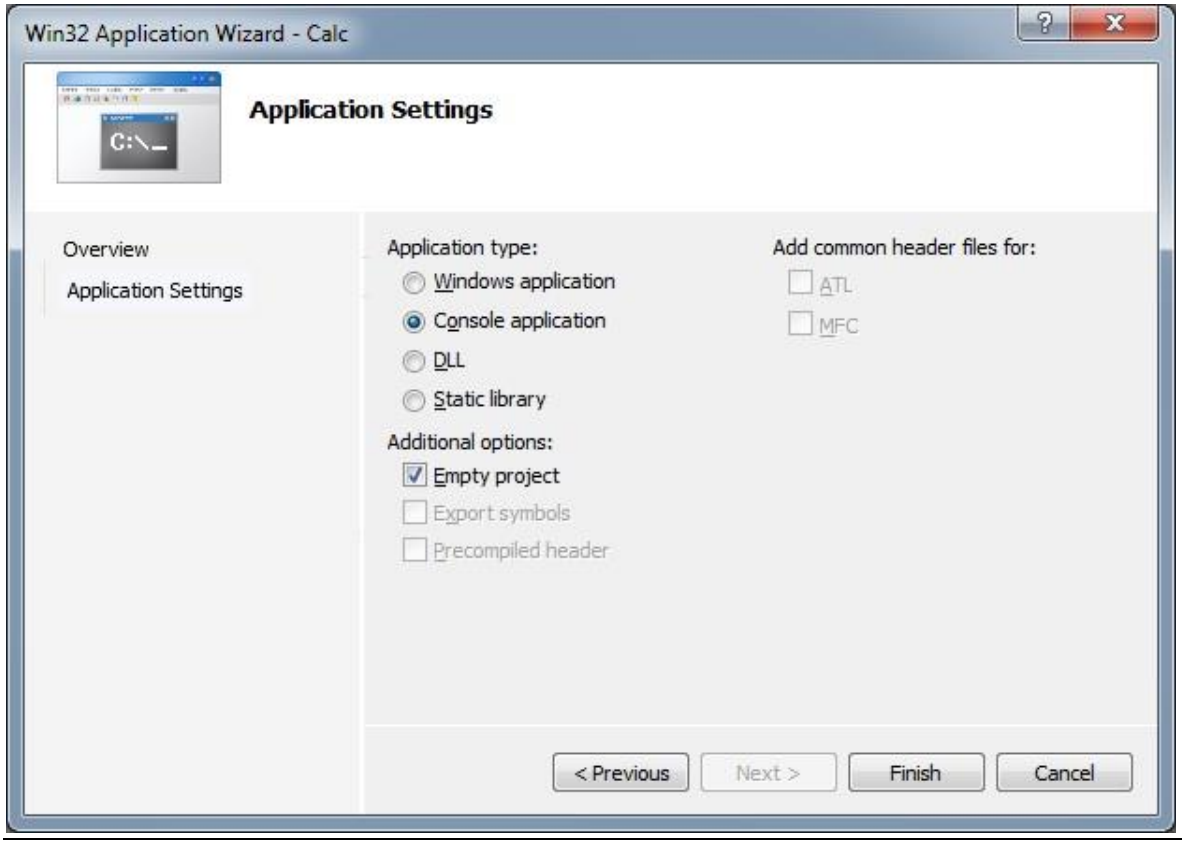
В появившейся карточке выбираем тип проекта «консольное приложение», то есть – приложение текстового режима, ввод данных в которое будет осуществляться с клавиатуры, а вывод – в текстовое окно на экране компьютера.

В этом же окне нужно задать имя проекта (для первой программы – Calc) и место на дисках, где проект будет расположен. Вместо диска C: здесь следует указать домашний диск студента в практикуме (обычно – диск Z:).



2.4 Application settings -> Application settings

Вслед за нажатием кнопки ОК в предыдущем диалоге появится окно настроек свойств будущего проекта. Точнее, будет раскрыто окно с описанием свойств проекта «по умолчанию», но нужно в этом же диалоге переключиться на пункт «Настройки» приложения (Application settings) и установить все органы управления так, как это показано на приводимой ниже иллюстрации:

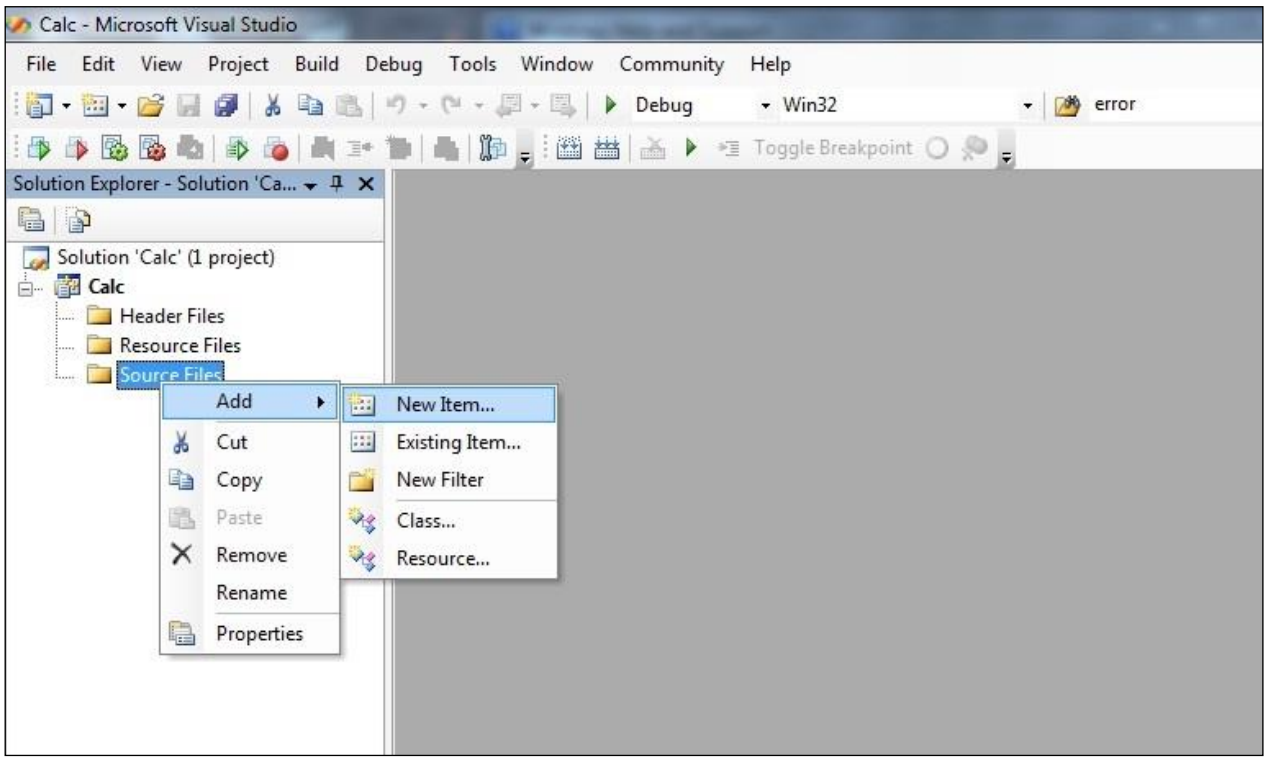


Именно здесь мы задаем тип проекта «консольное приложение» (Console application), а также отменяем использование пред-компилированных заголовочных файлов языка Си (для небольших учебных программ выигрыш в скорости сборки от них невелик, а вот затруднения с перекомпиляцией приложения у студентов возникают часто).

Последняя опция, которую надо задать здесь: «Пустой проект» (Empty project), означает, что проект будет создан пустым, без автоматической генерации шаблона приложения и других сопроводительных файлов. Делается это для того, чтобы студент не отвлекался на файлы, смысл которых ему пока будет непонятен.

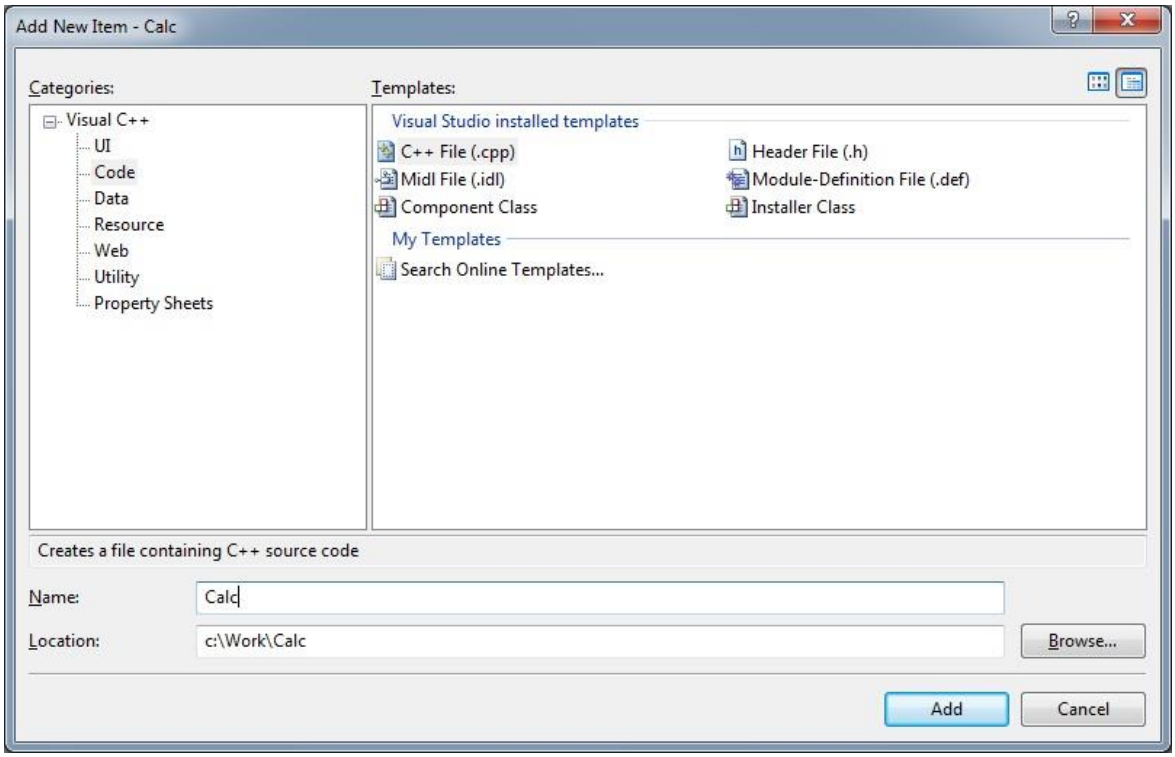
2.5 Project -> Add new item -> C++ source file (.cpp)

После выполнения предыдущих действий пустой проект будет заведен и в его папку «Исходные тексты» (Source Files) нужно добавить текстовый файл, который будет содержать текст будущей программы. Это можно сделать как через меню Project, так и щелчком правой кнопки мыши по папке Source Files созданного проекта.



В появившемся затем диалоге нужно указать тип добавляемого в проект файла – исходный текст языка C++ (C++ File). Нужно пояснить, что это файл именно языка программирования C++, для которого изучаемый в первом семестре язык программирования Си является почти строгим подмножеством. Необходимо это потому, что удобно сразу освоить основные операторы ввода-вывода языка C++, без необходимости объяснять сложные параметры функций ввода-вывода языка Си.

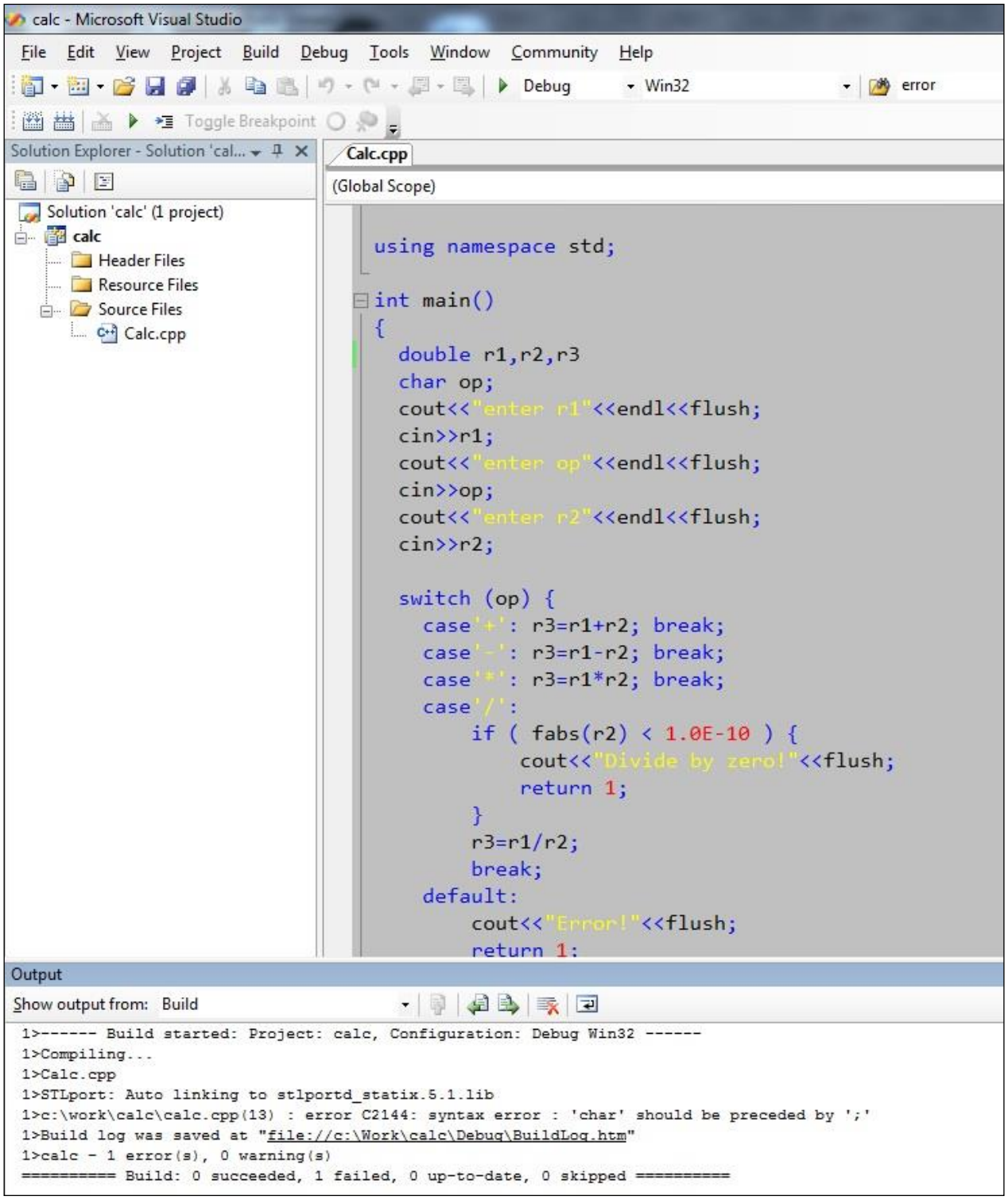
Естественно, нужно указать и имя файла (удобно, чтобы оно совпадало с именем проекта) и его месторасположение на диске компьютера (обычно – в папке, автоматически созданной для проекта при его заведении), расширение будет подставлено автоматически:



Рассмотрение прочих возможных компонент проекта выходит за рамки курса (они предназначены для графических приложений среды Microsoft Windows), можно только очень кратко описать назначение заголовочных файлов (Header File (.h)), которым будет посвящена тема на одном из следующих семинарских занятий.

2.6 Окончательный вид созданного проекта

На примере данной иллюстрации нужно пояснить основные панели (зоны) интерактивной среды программирования, их назначение и иерархию сущностей проекта (Solution – Project – Source Files).



Панель слева – Solution Explorer, в ней будут представлены все проекты, входящие в данное решение (Solution), для одиночного проекта одноименный файл решения (Solution) генерируется автоматически, но для более сложных задач может понадобится целый набор разнотипных компонент проекта (выполняемые файлы, динамические библиотеки и т.п.), которые будут зависеть друг от друга и должны собираться в строго определенной последовательности.

Правая панель отведена для текстового редактора, в котором происходит написание и исправление текста программы. Изначально этот файл (открываемый двойным щелчком мыши по имени файла в Solution Explorer) пуст, студент самостоятельно пишет в нем текст своей программы.

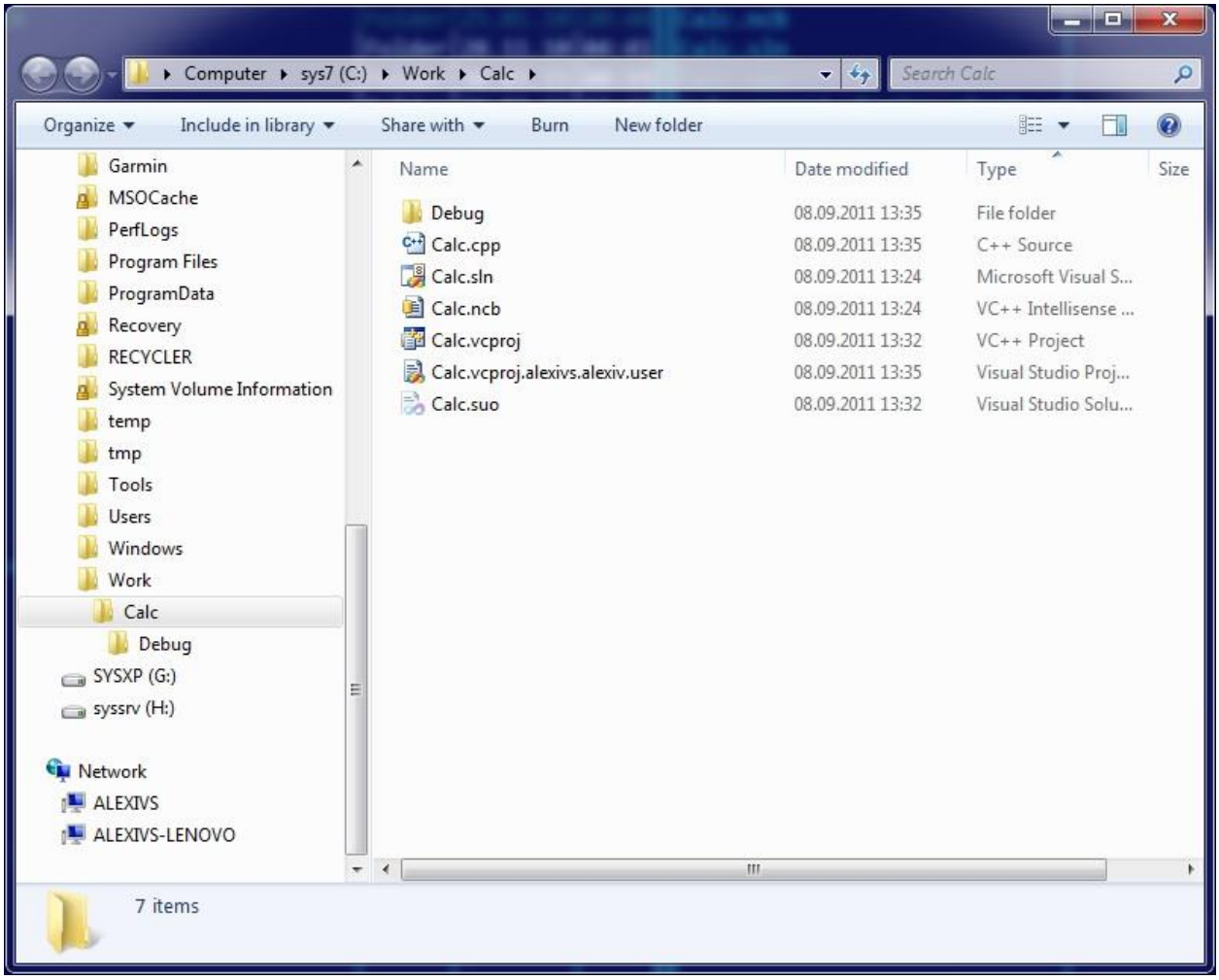
Панель снизу открывается автоматически при запуске сборки проекта (см. ниже). В эту панель выводятся все текстовые сообщения: сообщения об ошибках и об успешном окончании этапов сборки проекта (компиляция, линковка).

Если в этой панели выведено сообщение об ошибке компиляции – то нужно анализировать их сверху вниз, начиная с самой первой, так как последующие ошибки компиляции могут быть наведенными, вызванными предыдущими. Двойной щелчок мышью по строке с сообщением об ошибке позиционирует окно текстового редактора на строку, которая явилась причиной ошибки.

Если подвести текстовый курсор к номеру ошибки (`C2144` на иллюстрации выше) и нажать на клавиатуре клавишу `F1` – то справочная система интерактивной среды выдаст подробное описание данной ошибки и примеры ее возникновения.

2.7 Содержимое папок созданного и собранного проекта на диске

После создания проекта в рабочей папке образуется следующая файловая структура:



То есть, каждое решение (Solution) занимает отдельную папку, в которой расположены все файлы, необходимые для сборки всех проектов, входящих в данное решение:

- `Calc.cpp` – исходный текст написанной программы.

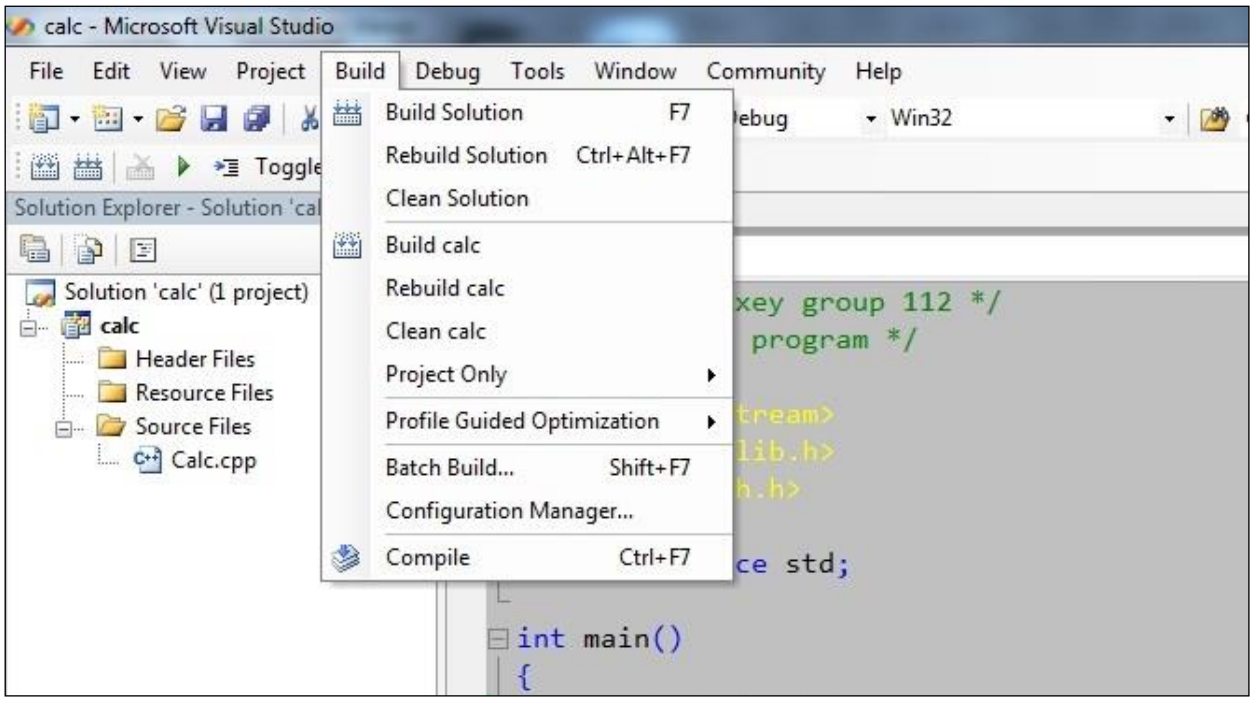
- `Calc.sln` – файл, описывающий состав решения (Solution), именно этот файл следует открывать при помощи команды File/Open интегрированной среды программирования.
- `Calc.vcproj` – файл проекта (Project), описывающий состав исходных текстов одной компоненты решения, в данном случае – исполняемой программы. Его можно включить в то или иное решение (и даже в несколько решений сразу), но для простых учебных программ обычно включаться будет единственный проект в единственное соответствующее проекту решение.

Прочие файлы этой папки являются служебными, создаются они средой программирования автоматически, останавливаться на их роли мы не будем.

Папка `Debug` содержит все результаты компиляции и сборки всех проектов решения, в частности – исполняемый файл нашей программы (`Calc.exe`). Папка `Debug` соответствует отладочному режиму сборки проекта, собираемые в ней компоненты будут содержать отладочную информацию и в них будет отключена часть оптимизаций программного кода, выполняемых компилятором.

Если в среде программирования выбрать режим сборки релизной версии программы (это делается переключателем в панели инструментов, расположенной прямо под меню интегрированной среды), то будет создана папка `Release`. Таким образом, компоненты отладочной и окончательной версии никогда не перепутаются, каждая версия программы будет собираться в отдельной папке.

2.8 Сборка проекта (Build)



По готовности исходного текста программы ее можно попробовать собрать. Полный набор команд для этого содержится в меню Build (Сборка), показанном на иллюстрации выше. Здесь же видны сочетания «горячих клавиш», которые можно нажимать для быстрого запуска той или иной команды:

- `Ctrl+F7` или `Build -> Compile` – скомпилировать текущий (открытый в редакторе) файл с исходным текстом программы, выдать ошибки компиляции (если будут).

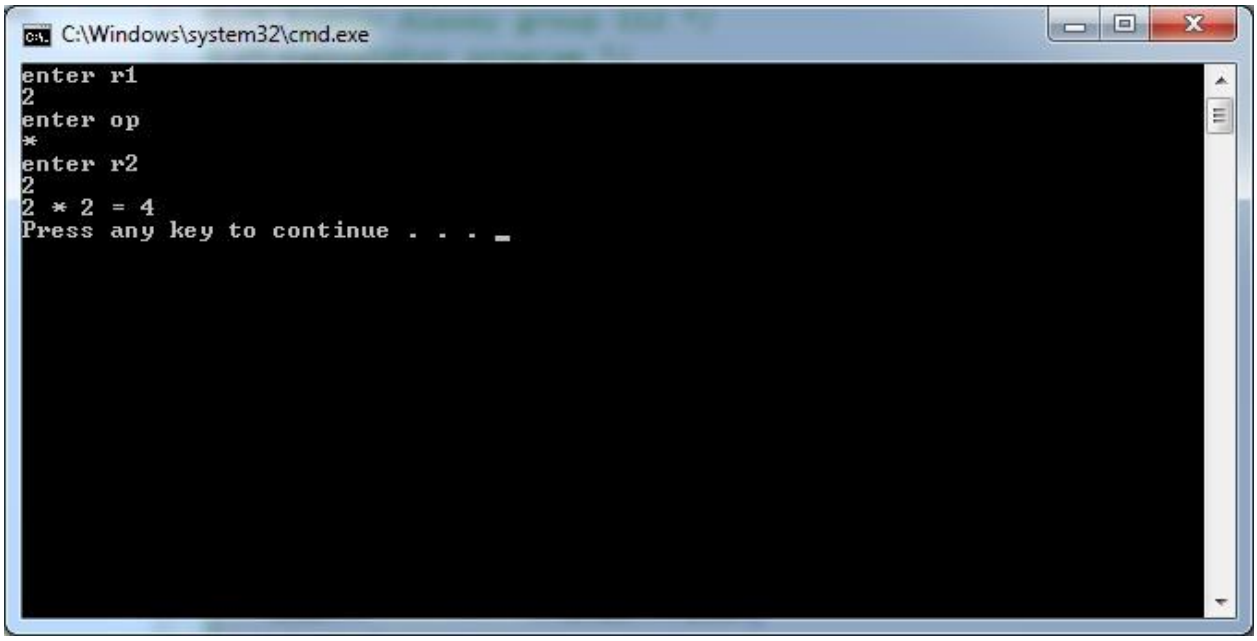
- `F7` или `Build -> Build Solution` – полностью собрать все решение, включая компиляцию всех исходных текстов и линковку всех проектов. При этом компилироваться и линковаться будут только измененные с последней сборки файлы, это очень экономит время сборки.
- `Ctrl+Alt+F7` или `Build -> Rebuild Solution` – полная пересборка решения: перекомпилируются абсолютно все файлы, составляющие проекты решения. Иногда помогает устранить непонятные ошибки компиляции, вызываемые сбоями сетевого оборудования.
- `Ctrl+F5` или `Debug -> Start Without Debugging` – запустить собранную программу на выполнение без отладчика.
- `F5` или `Debug -> Start Debugging` – запустить собранную программу на выполнение под отладчиком.

Запуск под отладчиком без указания точки остановки – приведет к быстрому мельканию и автоматическому закрытию окна с результатами программы, если программа не ожидает ввода от пользователя. При обычном запуске, в конце работы программы будет выведена надпись, предлагающая нажать любую клавишу для закрытия окна, то есть, пользователь сам сможет решить, когда закрывать окно.

Все команды доступны не только через главное меню интегрированной среды, но и с панели инструментов. Наведение указателя мыши на ту или иную кнопку панели инструментов раскрывает подсказку по смыслу данной команды и ее горячей клавише.

2.9 Результат выполнения первой программы

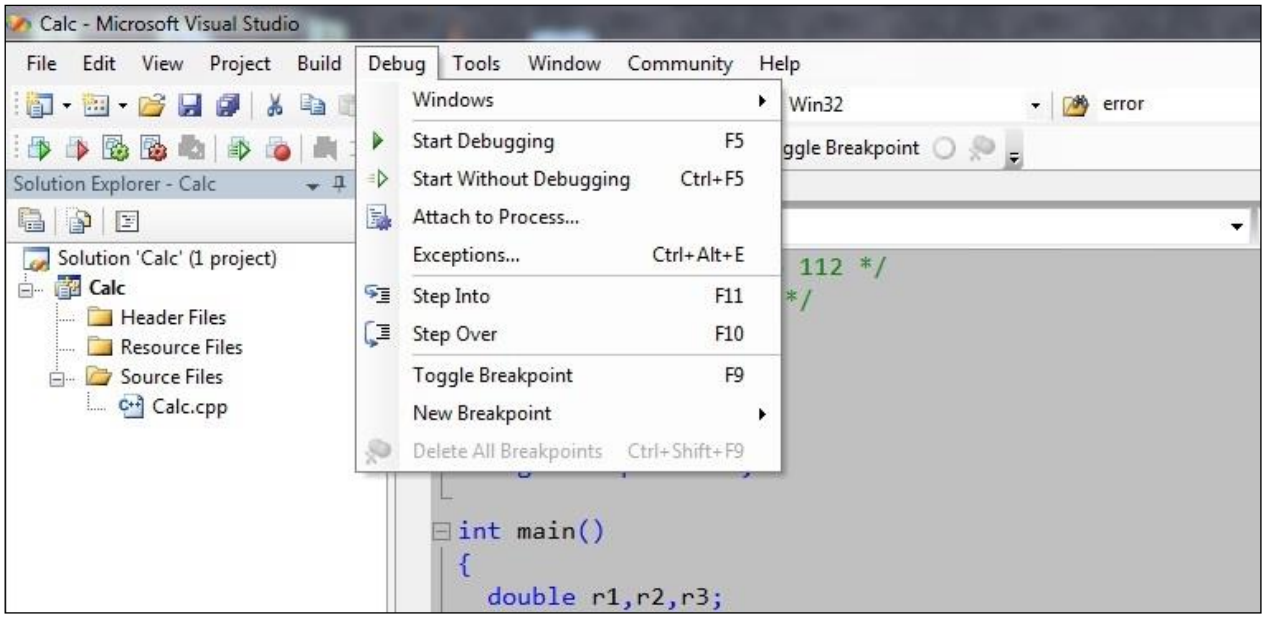
В результате сборки и запуска первой программы должно появиться примерно такое окно:



Здесь сначала пользователя предлагают ввести тот или иной член выражения (`enter r1`), затем он его печатает на клавиатуре и этот член появляется на экране (`2`). В конце программа печатает ответ выражения (`2 * 2 = 4`), после чего операционная система печатает свое приглашение нажать любую клавишу, чтобы закрыть окно (`Press any key to continue…`).

2.10 Отладка (Debug)

Интерактивный отладчик помогает найти ошибки в логике работы программы, которые невозможно обнаружить компилятору (компилятор может обнаружить только синтаксические ошибки).



Команды отладчика:

- `F11` или `Debug -> Step Into` – пошаговое выполнение программы, при этом каждая строчка исходного кода выполняется по нажатию клавиши `F11`, после чего курсор перемещается к следующей строке, а у программиста появляется возможность просмотреть значения всех переменных программы к этому моменту (например, просто подведя к ним курсор мыши или в нижней панели среды), а при необходимости можно даже исправить эти значения.

- `F10` или `Debug -> Step Over` – то же самое, что предыдущая команда (пошаговый режим), но, в отличие от нее, если в текущей строке есть вызов функции, то не происходит переход пошагового режима в тело этой функции. Функция просто вычисляется полностью, возвращается в точку вызова, после чего полностью выполняется текущая строка и отладчик переходит к следующей строке.

- `Shift+F9` или `Ctrl+Alt+Q` или `Debug -> Quick Watch` – быстрый просмотр значения переменной. Команда в меню появляется только, когда программа запущена под отладчиком! Нужно подвести курсор к интересующей переменной, нажать «горячие клавиши», после чего появится окно, в которой можно будет просмотреть значение переменной.

- `F9` или `Debug -> Toggle Breakpoint` – переключатель, позволяющий поставить или убрать точку остановки отладчика (Breakpoint) на текущей строке. Когда отладчик запущен командой F5 он будет быстро выполнять все строки программы до тех пор, пока не дойдет до точки остановки, в которой отладчик остановится. В точке остановки можно просмотреть значения переменных и постараться понять – правильно работает программа или нет.

- `F5` или `Debug -> Start Debugging` – продолжить выполнение программы под отладчиком после точки остановки (до конца программы или до следующей точки остановки, которых может быть сколько угодно).

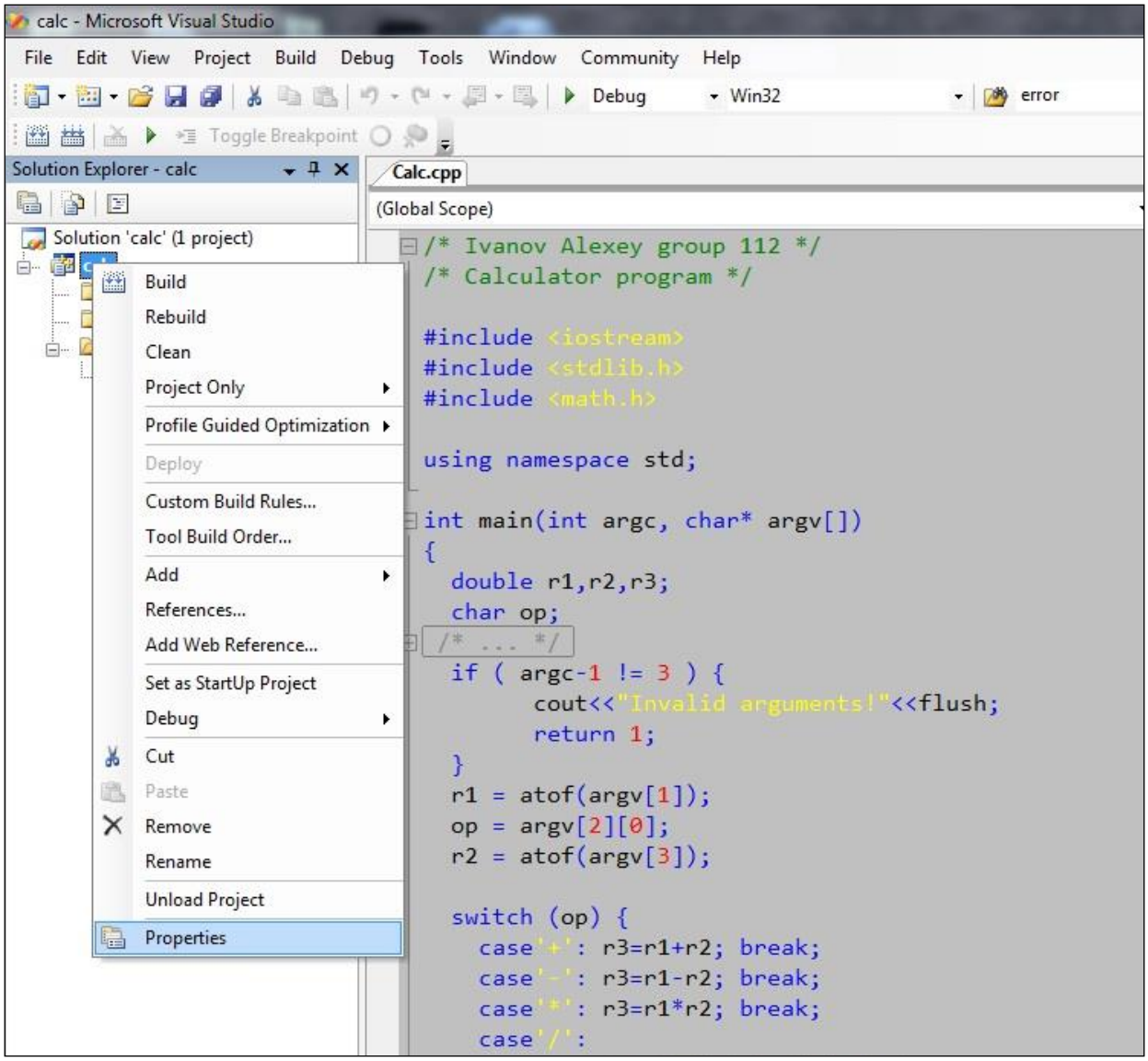
- `Shift+F5 Stop Debugging` – прервать выполнение программы и ее отладку. Производится возврат к редактированию исходного текста программы.

Не рекомендуется вносить изменения в исходный текст программы во время выполнения ее под отладчиком. Лучше прервать отладку, исправить и перекомпилировать программу, после чего запустить отладку заново.

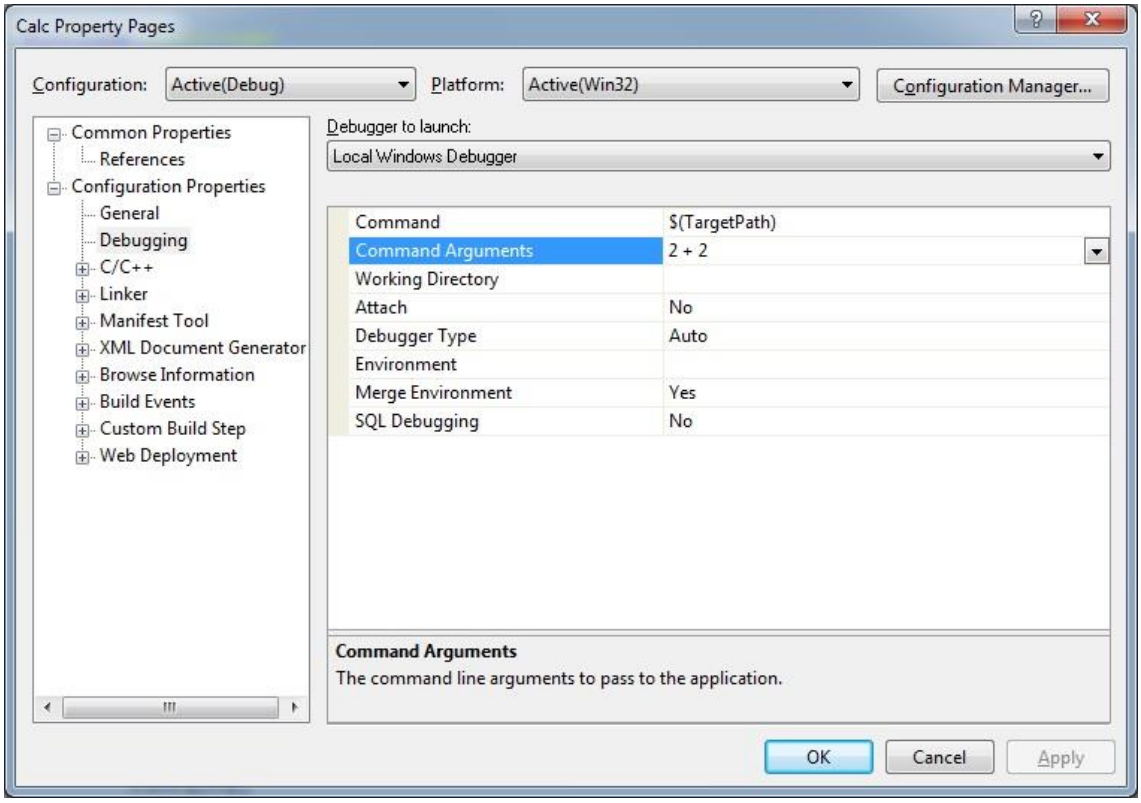
2.11 Задание параметров для командной строки программы в интерактивной среде

Иногда бывает удобно задать необходимые аргументы для выполнения программы прямо в ее командной строке при запуске. В таком случае, аргументы следует указать непосредственно в среде программирования, любой последующий запуск программы (под отладчиком или нет) будет использовать именно указанные аргументы.

Для задания аргументов программы следует щелкнуть правой кнопкой мыши по проекту и вызвать редактор свойств проекта:



В диалоге редактирования свойств проекта следует выбрать `Configuration Properties -> Debugging`, а потом, в правой панели задать аргументы командной строки (`Command Arguments`) с пробелами между отдельными параметрами программы:



3 Первая программа

Первая программа, которую студенты должны написать в практикуме, представляет из себя простейший калькулятор для вводимых пользователем чисел.

Важный момент: в данной программе используются операторы ввода-вывода языка C++, в силу того, что они гораздо легче осваиваются студентами на начальных этапах обучения, по сравнению с функциями ввода-вывода `scanf()/printf()`. Тем не менее, использование функций ввода-вывода языка Си можно оставить на усмотрение преподавателя.

```
/* Ivanov Alexey group 112 */
/* Calculator program */

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

using namespace std;

int main()
{
  double r1,r2,r3;
  char op;
  cout<<"enter r1"<<endl<<flush;
  cin>>r1;
  cout<<"enter op"<<endl<<flush;
  cin>>op;
  cout<<"enter r2"<<endl<<flush;
```

```
  cin>>r2;

  switch (op) {
    case'+': r3=r1+r2; break;
    case'-': r3=r1-r2; break;
    case'*': r3=r1*r2; break;
    case'/':
        if ( fabs(r2) < 1.0E-10 ) {
            cout<<"Divide by zero!"<<flush;
            return 1;
        }
        r3=r1/r2;
        break;
    default:
        cout<<"Error!"<<flush;
        return 1;
  }

  cout<<r1<<' '<<op<<' '<<r2<<" = "<<r3<<endl<<flush;
  return 0;
}
```

Небольшая модификация программы позволит пользователю вводить аргументы выражения не с консоли, а непосредственно в командную строку программы:

```
…
int main(int argc, char* argv[])
{
  double r1,r2,r3;
  char op;

  if ( argc-1 != 3 ) {
        cout<<"Invalid arguments!"<<flush;
        return 1;
  }
  r1 = atof(argv[1]);
  op = argv[2][0];
  r2 = atof(argv[3]);

  switch (op) {
    case'+': r3=r1+r2; break;
...
```

Здесь функция `atof()` производит преобразование строкового типа (массив типа `char`) в соответствующее вещественное значение. Для получения целых значений есть аналогичная функция `atoi()`, обе – входят в системную библиотеку `<stdlib.h>`.

Запустить такую программу можно будет либо прямо из интегрированной среды, как это было описано в предыдущем разделе, либо из программы `Start -> Accessories -> Command Prompt` так:

```
...\Debug> calc.exe 2 + 2
...
```

Пробелы между аргументами в данном случае важны.

Для того, чтобы программа могла выводить в консоль тексты и приглашения на русском языке рекомендуется вставить до первого оператора вывода вызов следующей системной функции:

```
  setlocale(LC_ALL,"rus");
```

К сожалению, обеспечить ввод строк на русском языке таким образом нельзя, это гораздо сложнее, но для учебных программ первого семестра – это и не требуется.

Типовое задание: написать и протестировать работу программы-калькулятора, выполняющую арифметические действия с числами, вводимыми пользователем. Добиться ее работоспособности. Это задание должны выполнить все студенты группы, только после его сдачи можно давать дополнительные задания по образцу приводимых ниже.

В практикуме нужно объяснить и отработать элементарные действия в отладчике программ: пошаговое исполнение, контроль значений переменных, точки останова.

При сдаче задания студента нужно попросить модифицировать программу: добавить обработку ошибок, добавить новые действия (возведение в степень, тригонометрия), добавить количество выполняемых действий (несколько операций, но без обработки приоритетов и скобок).

В качестве дополнительных заданий успевающим студентам можно предложить более серьезную модификацию этой программы, например, организацию ввода выражений прямо в командной строке или организацию «бесконечного» цикла вычислений (вычисление многих выражений, пока пользователь не выдаст команду на завершение работы).

1. Вариант

Дополнить **калькулятор** возможностью ввода аргументов через параметры командной строки.

При заданных аргументах в командной строке должны использоваться именно эти аргументы, при недостатке каких-либо аргументов, в том числе – при отсутствии всех аргументов командной строки, недостающие аргументы запросить у пользователя.

2. Вариант

Дополнить **калькулятор** возможностью ввода с клавиатуры неограниченного количества выражений (до ввода выражения `0 + 0`).

Требуется либо использовать цикл, либо использовать метку и оператор `goto` в совокупности с условным оператором.

3. Вариант

Дополнить **калькулятор** возможностью ввода либо одной, либо двух операций.

Нужно, чтобы можно было вычислять выражения вида: `2 + 6 / 4` наряду с выражениями вида: `3 - 1`. Приоритет операций можно не учитывать, вычислять выражение слева направо, т.е. `(2+6)/4=2`.

4. Вариант

Дополнить **калькулятор** возможностью ввода двух операций с учетом приоритета умножения и деления над сложением и вычитанием.

Программа должна правильно обрабатывать случаи вида: `4+8/4=6` и вида: `8/4-3=-1`.

5. Вариант

Написать калькулятор, вычисляющий **значение полинома в заданной точке**.

Коэффициенты полинома задаются в командной строке (от нулевой до четверной степени), координаты точки – запрашиваются у пользователя.

6. Вариант

Дополнить **калькулятор** дополнительной операцией – **вычислением процентов**.

Наряду с четырьмя арифметическими действиями калькулятор должен вычислять выражения вида: `8%4=50`, то есть, сколько процентов второй аргумент составляет от первого.

7. Вариант

Дополнить **калькулятор** дополнительной операцией – **возведением в целую степень** не выше четвертой.

Наряду с четырьмя арифметическими действиями калькулятор должен вычислять выражения вида: `2^4=16`, при этом потребуется использовать циклы или оператор `goto`. Системную функцию не использовать! При нецелом втором аргументе допускается его округлять вниз.

8. Вариант

Дополнить **калькулятор** дополнительной операцией – **возведением в степень** (любую).

Наряду с четырьмя арифметическими действиями калькулятор должен вычислять выражения вида: `2^4=16`, при этом можно воспользоваться системной функцией `pow(x,y)`.

9. Вариант

Дополнить **калькулятор** дополнительной операцией – **вычислением синуса**.

Наряду с четырьмя арифметическими действиями калькулятор должен вычислять выражения вида: `sin 1.5`, лучше, если с обработкой скобок: `sin(1.5)` и выдачей сообщения об ошибке, если со скобками что-то не так. Использовать системную функцию `sin(x)`.

10. Вариант

Написать калькулятор, вычисляющий **функцию колебаний** $x = A \sin(\omega t+\varphi)$.

Калькулятор должен вычислять выражения вида: `3 A 1 O 2 P`, где `3 A` означает «амплитуда колебаний равна трем, `1 O` означает «частота колебаний равна единице», а `2 P` означает «фаза равна двум». Эти аргументы задаются в командной строке, а значение времени – запрашивается у пользователя.

11. Вариант

Написать калькулятор, выполняющий **линейную интерполяцию** между двух заданных точек на плоскости.

Калькулятор должен вычислять выражения вида: `1.1 x1 1 y1 2 x2 1.5 y2`, где `3 x1` означает «x координата первой точки равна 1.1», `1 y1` означает «y координата первой точки равна 1», `2 x2` означает «x координата второй точки равна 2», а `1.5 y2` означает «y координата второй точки равна 1.5». Эти аргументы задаются в командной строке, а значение координаты x, в которой надо интерполировать значение – запрашивается у пользователя.

12. Вариант

Дополнить **калькулятор** дополнительной операцией – **обратным делением**.

Наряду с четырьмя арифметическими действиями калькулятор должен вычислять выражения вида: `2\12=6`, то есть операция должна делить правый операнд на левый (обратная косая черта символизирует черту дроби для этого случая).

13. Вариант

Написать калькулятор, вычисляющий квадратный **корень из факториала** введенной пользователем величины.

Калькулятор должен вычислять выражение вида: $\sqrt{x!}$ для любого введенного пользователем значения $0 \le x \le 4$, ввод отрицательного x должен давать ошибку, нецелые числа допускается округлять.

14. Вариант

Написать программу, вычисляющую **минимальное число**, из произвольного количества чисел, вводимых пользователем.

Числа указываются в командной строке в произвольном порядке, программа должна напечатать их и указать, какое по счету число меньше всех остальных. Нужно воспользоваться оператором цикла или операторами `if()` и `goto`.

15. Вариант

Написать программу, решающую **квадратное уравнение** $ax^2+bx+c=0$ для трех коэффициентов, вводимых пользователем.

Коэффициенты указываются в командной строке, программа должна напечатать уравнение и его решение, подставить найденное решение в уравнение и напечатать результат. Системная функция вычисления квадратного корня: `sqrt(x)`.

16. Вариант

Написать программу, вычисляющую **определитель** квадратной матрицы размерности 2x2.

Коэффициенты матрицы указываются в командной строке, программа должна напечатать матрицу и ее определитель.

17. Вариант

Дополнить **калькулятор** возможностью ввода двух операций **с обязательным указанием скобок** для уточнения приоритета операций.

Приоритет умножения и деления над сложением и вычитанием учитывать не обязательно. Программа должна правильно обрабатывать случаи вида: `4+8/4=3`, вида: `(4+8)/4=3`, и вида: `4+(8/4)=6`. При ошибках в употреблении скобок выдавать сообщение об ошибке.

18. Вариант

Дополнить **калькулятор** памятью промежуточных выражений и возможностью ввода с клавиатуры **неограниченного количества действий**, применяемых к полученному ранее итогу, вплоть до ввода операции деления на нуль.

Требуется либо использовать цикл, либо использовать метку и оператор `goto` в совокупности с условным оператором, так, чтобы пользовательская сессия выглядела примерно так:

```
R1 ?
3
Op ?
+
R2 ?
2
3+2=5
Op ?
-
R2 ?
8
5-8=-3 …
```

19. Вариант

Дополнить **калькулятор** возможностью вводить операнды **в двоичной системе счисления**.

Для любого операнда префикс «b» означает, что этот операнд задан в двоичной системе счисления (не более 4-х разрядов): `b1001+5=14`. Числа без префикса считаются десятичными. Результат печатать в десятичной системе счисления.

20. Вариант

Написать **калькулятор**, складывающий **восьмеричные числа**.

Восьмеричные числа – не более трех разрядов, все разряды задаются обязательно: 007+015=024. У пользователя запрашивается пара чисел, печатается ответ.

21. Вариант

Написать **калькулятор**, складывающий **шестнадцатеричные числа**.

Шестнадцатеричные числа – не более двух разрядов, представляются десятичными цифрами 0-9 и буквами A-F, соответствующими числам с 10 до 15. У пользователя запрашивается пара чисел, печатается ответ.

22. Вариант

Дополнить **калькулятор** возможностью вводить операнды **в восьмеричной системе счисления**.

Для любого операнда префикс «o» означает, что этот операнд задан в восьмеричной системе счисления (не более 2-х разрядов): `o10+6=14`. Числа без префикса считаются десятичными. Результат печатать в десятичной системе счисления.

23. Вариант

Дополнить **калькулятор** возможностью вводить операнды **в шестнадцатеричной системе счисления**.

Для любого операнда префикс «x» означает, что этот операнд задан в шестнадцатеричной системе счисления (не более двух разрядов): `xFF+6=261`. Числа без префикса считаются десятичными. Результат печатать в десятичной системе счисления.

24. Вариант

Написать **калькулятор**, вычитающий **шестнадцатеричные числа**.

Шестнадцатеричные числа – не более двух разрядов, представляются десятичными цифрами 0-9 и буквами A-F, соответствующими числам с 10 до 15. У пользователя запрашивается пара чисел, печатается ответ.

25. Вариант

Написать **калькулятор**, складывающий и вычитающий **двоичные числа**.

Двоичные числа – не более четырех разрядов, представляются десятичными цифрами 0 и 1. У пользователя запрашивается пара двоичных чисел, печатается ответ тоже в двоичной форме.