# Лабораторная работа №2
# «Интегрированная среда для языков C/C++(IDE)»

## *1 Создание проекта в IDE.*

## *2 Анализируем и добавляем нужное.*

### Анализируем

1. Строки, начинающиеся с «//» — это комментарии. Компилятором они игнорируются (но бывают порой весьма полезны для человека, изучающего исходный код программы).
2. "`#include`" — это специальная директива, предназначенная для подключения к программе внешних модулей. В кавычках ("") подключаются файлы из текущей директории, в угловых скобках (<>) — модули стандартной библиотеки. **Мы пока будем использовать только второй вариант.**
3. А вот с этой языковой конструкции начинает выполняться любая программа:

```
int main(){
     // тут мы пишем собственно код программы
}
```

   Она означает:
   - определить **функцию** (функция — это фрагмент кода, который может быть *вызван*, в результате чего будет *выполнен*, и после завершения *вернет* число — результат своей работы; классический пример — функция sin, вычисляющая синус некоторого *аргумента*: sin(x));
   - имя функции — main;
   - функция не имеет аргументов (о чем свидетельствуют пустые круглые скобки);
   - в качестве результата своей работы функция обязана вернуть **целое число** (о чем говорит «тип возврата» — **int** (от англ. integer), см. ниже п.5).

   **Функция main является главной в любой программе и вызывается автоматически операционной системой при запуске программы.**
4. Все, что написано между открывающейся ("{") и закрывающейся ("}") фигурными скобками составляет **тело функции main**. **Именно оно и является, по сути, телом всей программы** (о том, что в нем писать, см. ниже).
5. "return 0;" — это оператор внутри функции main, который, собственно, означает: «завершить выполнение функции и вернуть в качестве результата число 0». Возврат именно нуля свидетельствует об успешном завершении вашей программы (в случае аварийного завершения принято возвращать код ошибки).

### Добавляем

Любая программа должна откуда-то получать исходные данные и куда-то записывать результаты своей работы над этими данными. Наши программы для этого будут использовать экран и клавиатуру. Соответствующая функциональность обеспечивается модулем **iostream** стандартной библиотеки. А как подключать их, мы уже знаем:

1. В начале программы, перед функцией **main**, напишите:

```
#include <iostream>
```

2. Сразу после этого (но все еще до main):

```
using namespace std;
```

Эта «волшебная» директива осуществляет включение идентификаторов из стандартной библиотеки в текущее пространство имен (“name space”) — в общем, теперь в нашей программе доступны стандартные потоки ввода и вывода `cin` и `cout` (см. ниже).

## *3 Компилируем и запускаем*

Команда «построить проект» — запускает процесс генерации *.exe-файла вашей программы. При этом внизу экрана, в окне “output” отображается ход процесса построения.

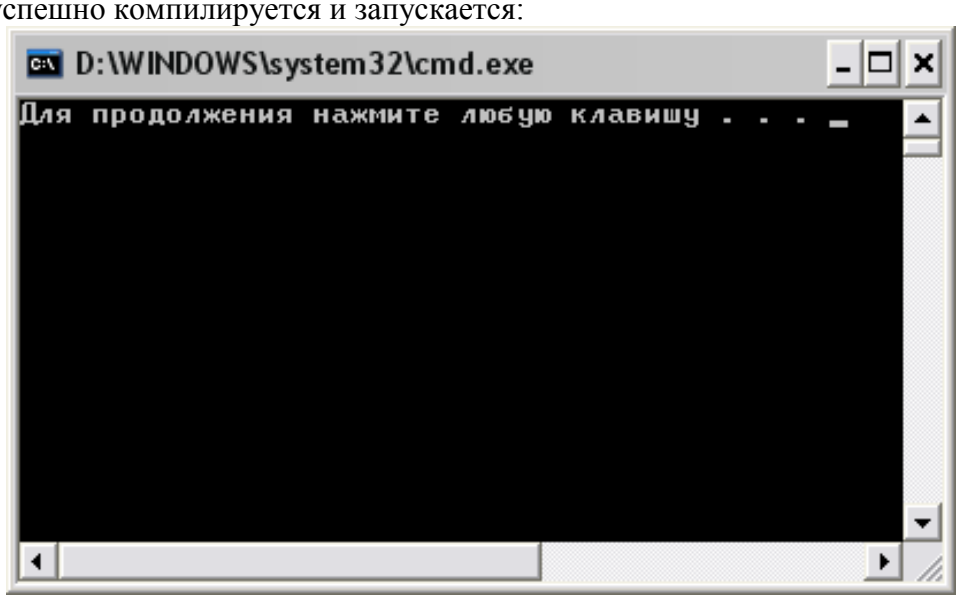
**По завершении этого процесса вы должны увидеть:**

```
Linking...
LINK : c:\dima\first\Debug\first.exe not found or not built by the last increm
Embedding manifest...
Microsoft (R) Windows (R) Resource Compiler Version 6.1.6723.1
Copyright (C) Microsoft Corporation.  All rights reserved.
Build log was saved at "file://c:\dima\first\first\Debug\BuildLog.htm"
first - 0 error(s), 0 warning(s)
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

После чего выполнить команду  — запуск программы под отладчиком, или
 — запуск программы без отладчика.

**Необходимо, чтобы по завершении вашей программы окно с результатами ее работы осталось открытым.**

В результате должна получиться программа, которая, хоть пока ничего и не делает полезного, но успешно компилируется и запускается:

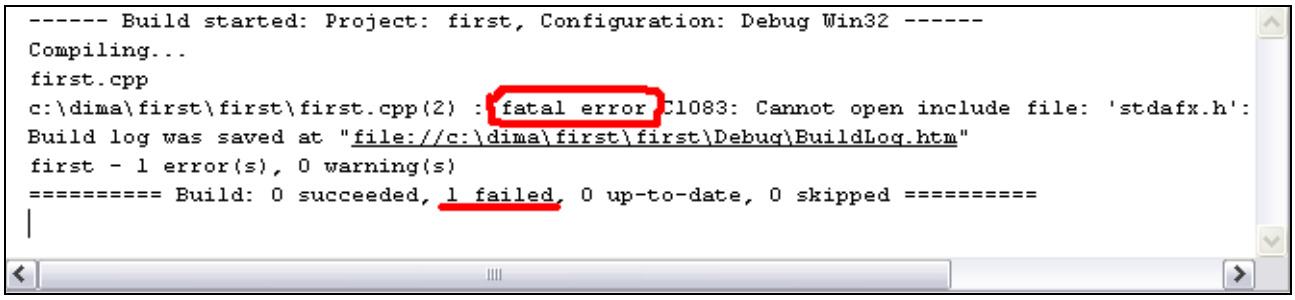



## *4 Если не компилируется*

Редкая программа скомпилируется (да еще и правильно заработает!) с первой попытки. Поэтому внимательно следите за сообщениями компилятора в окне «output». Сообщения в этом окне бывают двух типов:

- **Warning** — предупреждение. Не препятствует успешной компиляции программы, но может служить индикатором наличия потенциально небезопасного кода, который еще проявит себя на этапе выполнения.
- **Error** — ошибка, в результате которой дальнейшая компиляция и создание .exe-файла невозможно. Требует анализа и исправления.

Вот наглядный пример:




Довольно часто одна ошибка может стать причиной еще нескольких, т.к. делает дальнейший код совершенно «неадекватным». Поэтому первое правило исправления ошибок компиляции:

**Исправляйте ошибки в порядке их следования в окне «output» (т.е. начиная с самой верхней)!**

Далее, найдя первую по порядку ошибку, **кликните по ней два раза мышкой** — редактор автоматически поставит курсор на строку исходного кода, в которой обнаружена эта ошибка (также часто бывает полезно взглянуть на предыдущую строку). Оцените синтаксическую корректность кода, сравнив его, например, с образцом из учебника, конспекта или справочника по языку. После внесения изменений в код программу необходимо перекомпилировать .

Если код кажется синтаксически корректным, но компилятор все еще считает его ошибочным — внимательно прочитайте и переведите на русский язык **само сообщение об ошибке**. Дополнительную информацию об ошибке можно получить в MSDN Library по нажатии клавиши F1, когда курсор стоит на сообщении об этой ошибке.

## Примеры простых программ

Когда «тривиальная» программа из предыдущего раздела успешно скомпилировалась и запустилась (постарайтесь, чтобы warning'ов тоже не было) можно перейти к написанию простой программы, которая производит какие-нибудь действия. Наша первоочередная задача — освоить средства *ввода-вывода*, чтобы программы, которые мы будем писать в дальнейшем, могли откуда-то брать исходные данные для обработки и куда-то выводить результат (в нашем случае это будут клавиатура и экран соответственно).

**Задание 1. Наберите и заставьте работать программу с рис. 1.2 (с. 49), а также программы с рис. 1.4 и 1.5 (с. 52) книги Дейтела.**

**Внимание!** *Не надо полностью заменять свою теперешнюю программу на ту, что написана у Дейтела!* У вас **уже есть** `include`, `main`, `using` и прочее (и все это проверено и работает!). Просто добавьте в свою программу те строки, которые есть у Дейтела, но отсутствуют у вас!

Пояснение: во время написания книги Дейтела C++ немного отличался от современного, и поэтому некоторые «служебные» вещи у него написаны по-другому. Какие именно — можно легко понять, если сравнить рис. 1.2 книги с той программой, которая у вас получилась на данный момент.

**Задание 2. Наберите программу с рис. 1.6 (с. 53) и разберитесь, как она работает, по пояснениям раздела 1.16 книги.** Комментарии, как и раньше, перепечатывать не обязательно.

## Задания для самостоятельного решения

**Задание 3.** Изучите главу 1 книги Дейтела до конца. Ответьте на вопросы упражнений в конце главы.

**Задание 4.** Что будет выведено на экран при выполнении следующих операторов? Предполагайте при этом, что x=2, y=3. Если ничего не будет выведено, поставьте прочерк.

| Оператор | Результат на экране |
|---|---|
| `cout << x;` | |
| `cout << x + x;` | |
| `cout << "x=";` | |
| `cout << x" =" << x;` | |
| `cout << x + y << " = " << y + x;` | |
| `z = x + y;` | |
| `cin >> x >> y;` | |
| `// cout << "x + y = " << x + y;` | |
| `cout << "\n";` | |

**Задание 5.** Напишите программу, которая производит следующие действия:

a) Объявить переменные `x`, `y`, `z` и `result` типа `int`.
b) Предложить пользователю ввести три целых числа.
c) Прочитать три целых числа с клавиатуры и сохранить их в переменных `x`, `y` и `z`.
d) Вычислить произведение трех целых чисел, содержащихся в переменных `x`, `y`, `z` и присвоить результат переменной `result`.
e) Напечатать «Result is » и затем значение переменной `result`.
f) Возвратить из функции `main` значение, свидетельствующее об успешном завершении программы.

**Задание 6.** Операции "/" и "%" при применении их к целым числам обозначают в C++ соответственно *целочисленное деление* и *взятие остатка от деления*. Напишите программу, которая заполнит и выведет на экран таблицу следующего вида (заполнить столбцы «a/b» и «a%b»):

```
a  b  a/b  a%b
0  3
1  3
2  3
3  3
4  3
5  3
6  3
```

**Задание 7.** Напишите программу, которая считывает с клавиатуры трехзначное целое число и печатает на экране цифры этого числа «в столбик». Например, если введено число 123, на экране мы увидим:

```
1
2
3
```

Подсказка: 123 / 100 = 1, 123 / 10 % 10 = 2, 123 % 10 = 3.

## Контрольные вопросы

1. Какое значение хранит переменная x, и какое — переменная y после выполнения следующего кода?

```
int x, y;
x = 2; y = 3;
y = x;
x = 5;
```

2. Что является результатом работы компилятора и что — компоновщика?
3. Где (в каком именно месте компьютера) хранятся переменные, используемые в вашей программе? Как долго они могут там храниться?
4. Какой из двух потоков: cin и cout — используется для ввода, а какой для вывода? Какой из них привязан к клавиатуре, а какой — к экрану?
5. Как изменится работа программы, если оператор return написать в самом начале функции main()?

# Оценивание

## *Содержание отчета*

1. Условия заданий.
2. Исходные коды и скриншоты выполнения программ из заданий 1, 2, 5, 6, 7.
3. Заполненная таблица из задания 4.

## *Баллы за задания*

<table>
<tr><td>Задание</td><td colspan="2">Баллы</td></tr>
<tr><td>1</td><td>обязательное</td><td rowspan="3">1</td></tr>
<tr><td>2</td><td>обязательное</td></tr>
<tr><td>3</td><td>обязательное</td></tr>
<tr><td>4</td><td colspan="2">0.5</td></tr>
<tr><td>5</td><td colspan="2">0.5</td></tr>
<tr><td>6</td><td colspan="2">0.5</td></tr>
<tr><td>7</td><td colspan="2">1</td></tr>
<tr><td>Всего</td><td colspan="2">3.5</td></tr>
</table>

## *Бонусы*

Досрочная сдача: +0.5 балла.
Несвоевременная сдача: макс. балл уменьшается на 0.5 за каждую просроченную неделю.