

Язык и среда

Язык программирования — формальный язык, предназначенный для записи компьютерных программ. Язык программирования определяет набор лексических, синтаксических и семантических правил, определяющих внешний вид программы и действия, которые выполнит исполнитель под её управлением.

Три составляющих языка:

Алфавит, синтаксис и семантика.

Алфавит: конечное множество допустимых атомарных (неделимых) символов какого-либо формального языка, используемых для записи программы

Синтаксис языка программирования: набор правил, описывающий комбинации символов алфавита, считающиеся правильно структурированной программой (документом) или её фрагментом. Синтаксису языка противопоставляется его семантика. Синтаксис языка описывает «чистый» язык, в то же время семантика приписывает значения (действия) различным синтаксическим конструкциям.

Семантика языка — это смысловое значение слов. В программировании — начальное смысловое значение операторов, основных конструкций языка и т. п.

Интересно:

Язык программирования определяется не только через спецификации стандарта языка, формально определяющие его синтаксис и семантику, но и через воплощения (реализации) стандарта: программные средства, обеспечивающие трансляцию и интерпретацию программ на этом языке. Такие программные средства различаются по производителю, марке и варианту (версии), времени выпуска, полноте воплощения стандарта, дополнительным возможностям; могут иметь определенные ошибки или особенности воплощения, влияющие на практику использования языка и даже на его стандарт

Интегрированная среда разработки, ИСР (англ. *Integrated development environment* — **IDE**), также **единая среда разработки, ЕСР** — комплекс программных средств, используемый программистами для разработки программного обеспечения (ПО).

Обычно среда разработки включает в себя:

- текстовый редактор,
- транслятор (компилятор и/или интерпретатор),
- средства автоматизации сборки,
- отладчик.

Трансля́тор — программа или техническое средство, выполняющее трансляцию программы.

Трансля́ция програ́ммы — преобразование программы, представленной на одном из языков программирования, в программу на другом языке. Транслятор обычно выполняет также диагностику ошибок, формирует словари идентификаторов, выдаёт для печати текст программы и т. д.

Цель трансляции — преобразование текста с одного языка на язык, понятный адресату. При трансляции компьютерной программы адресатом может быть:

- устройство — процессор (трансляция называется компиляцией);
- программа — интерпретатор (трансляция называется интерпретацией).

Виды трансляции:

- компиляция;
 - в исполняемый код
 - в машинный код
 - в байт-код
 - транспиляция;
- интерпретация.

Компиля́тор — программа или техническое средство, выполняющее компиляцию.

Компиля́ция — трансляция программы, составленной на исходном языке высокого уровня, в эквивалентную программу на низкоуровневом языке, близком машинному коду (абсолютный код, объектный модуль, иногда на язык ассемблера). Входной информацией для компилятора (исходный код) является описание алгоритма или программа на предметно-ориентированном языке, а на выходе компилятора — эквивалентное описание алгоритма на машинно-ориентированном языке (объектный код).

Интерпретация — процесс чтения и выполнения исходного кода. Реализуется программой — интерпретатором.

Интерпретатор может работать двумя способами:

1. читать код и исполнять его сразу (чистая интерпретация);
2. читать код, создавать в памяти промежуточное представление кода (байт-код или р-код), выполнять промежуточное представление кода (смешанная реализация).

В первом случае трансляция не используется, а во втором — используется трансляция исходного кода в промежуточный код.

Интересно:

Этапы работы интерпретатора:

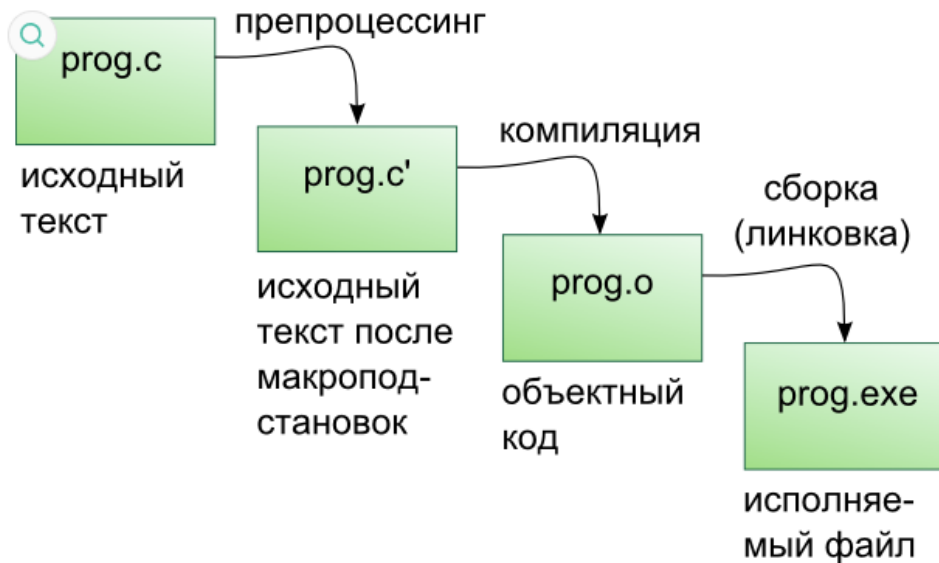
- лексический анализ;
- синтаксический анализ;
- семантический анализ;
- создание промежуточного представления кода (при чистой интерпретации не выполняется);
- исполнение.

Интерпретатор моделирует машину (виртуальную машину), реализует цикл выборки-исполнения команд машины. Команды машины записываются не на машинном языке, а на языке высокого уровня. Интерпретатор можно назвать исполнителем языка виртуальной машины.

Недостатки интерпретаторов по сравнению с компиляторами:

- низкая производительность (машинный код исполняется процессором, а интерпретируемый код — интерпретатором; машинный код самого интерпретатора исполняется процессором);
- необходимость наличия интерпретатора на устройстве, на котором планируется интерпретация программы;
- обнаружение ошибок синтаксиса на этапе выполнения (актуально для чистых интерпретаторов).

Процесс преобразования программы из исходного кода в исполняемый файл:



Компиляция

Препроцессинг

Эту операцию осуществляет текстовый препроцессор.

Исходный текст частично обрабатывается — производятся:

- Замена комментариев пустыми строками
- Текстовое включение файлов — `#include`
- Макроподстановки — `#define`
- Обработка директив условной компиляции — `#if`, `#ifdef`, `#elif`, `#else`, `#endif`

Процесс компиляции состоит из следующих этапов:

1. **Лексический анализ.** Последовательность символов исходного файла преобразуется в последовательность лексем.
2. **Синтаксический анализ.** Последовательность лексем преобразуется в дерево разбора.
3. **Семантический анализ.** Дерево разбора обрабатывается с целью установления его семантики (смысла) — например, привязка идентификаторов к их декларациям, типам, проверка совместимости, определение типов выражений и т. д.
4. **Оптимизация.** Выполняется удаление излишних конструкций и упрощение кода с сохранением его смысла.
5. **Генерация кода.** Из промежуточного представления порождается объектный код.

Результатом компиляции является **объектный код**. (*Файл OBJ!*)

Объектный код — это программа на языке машинных кодов с частичным сохранением символьной информации, необходимой в процессе сборки.

При **отладочной сборке** возможно сохранение большого количества символьной информации (идентификаторов переменных, функций, а также типов).

Компоновка

Также называется **связывание, сборка** или **линковка**.

Это последний этап процесса получения исполняемого файла, состоящий из **связывания воедино всех объектных файлов проекта**.

При этом возможны ошибки связывания.

Если, допустим, функция была объявлена, но не определена, ошибка обнаружится только на этом этапе.

Ошибки:

Синтаксические ошибки — это ошибки в записи конструкций языка программирования (чисел, переменных, функций, выражений, операторов, меток, подпрограмм).

Семантические ошибки — это ошибки, связанные с неправильным содержанием действий и использованием недопустимых значений величин.

Обнаружение большинства синтаксических ошибок автоматизировано в основных системах программирования. Поиск же семантических ошибок гораздо менее формализован; часть их проявляется при исполнении программы в нарушениях процесса автоматических вычислений и индицируется либо выдачей диагностических сообщений рабочей программы, либо отсутствием печати результатов из-за бесконечного повторения одной и той же части программы (зацикливания), либо появлением непредусмотренной формы или содержания печати результатов.

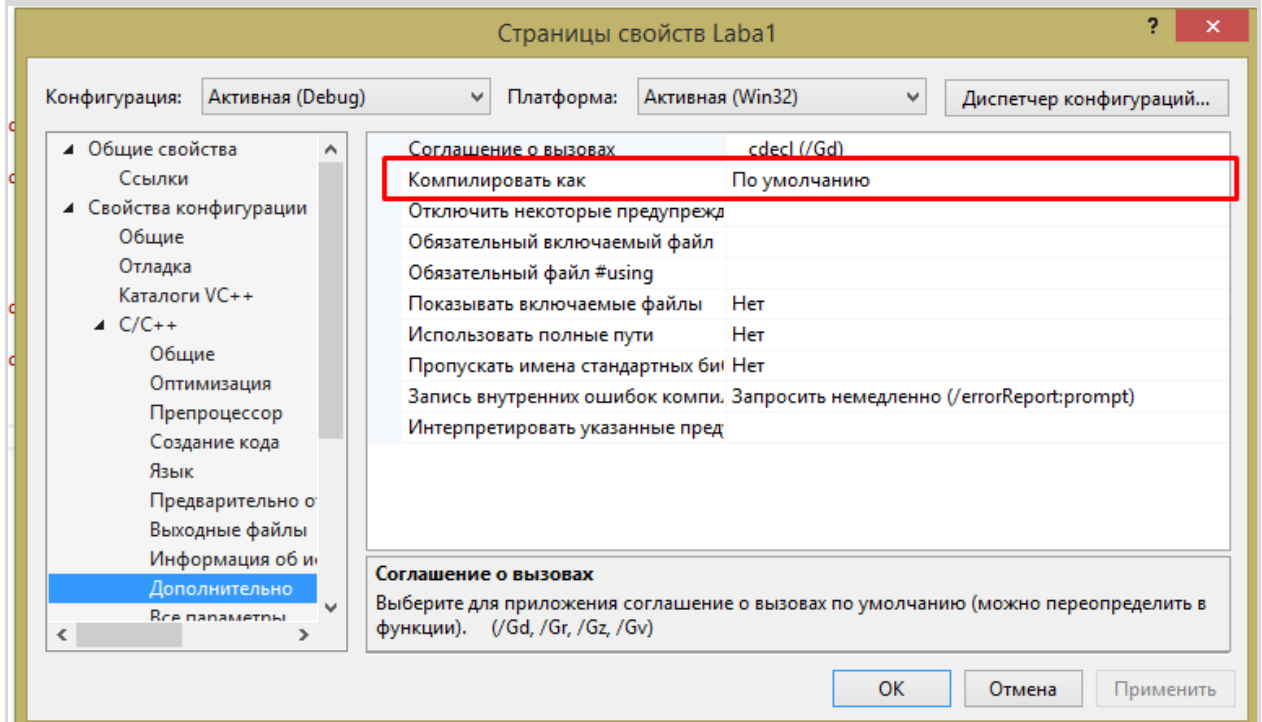
Начало работы со средой

Открыть среду->Создать проект->Проект->C++->Пустой проект

Файлы исходного кода-> Добавить-> Создать элемент-> Выбираем + меняем на .c

Формально этого достаточно, но можно еще иначе:

Проект->Свойства Имя_проекта -> "Свойства конфигурации"->"C/C++", -
>"Дополнительно"



Заменить выбранное значение на C(TC).

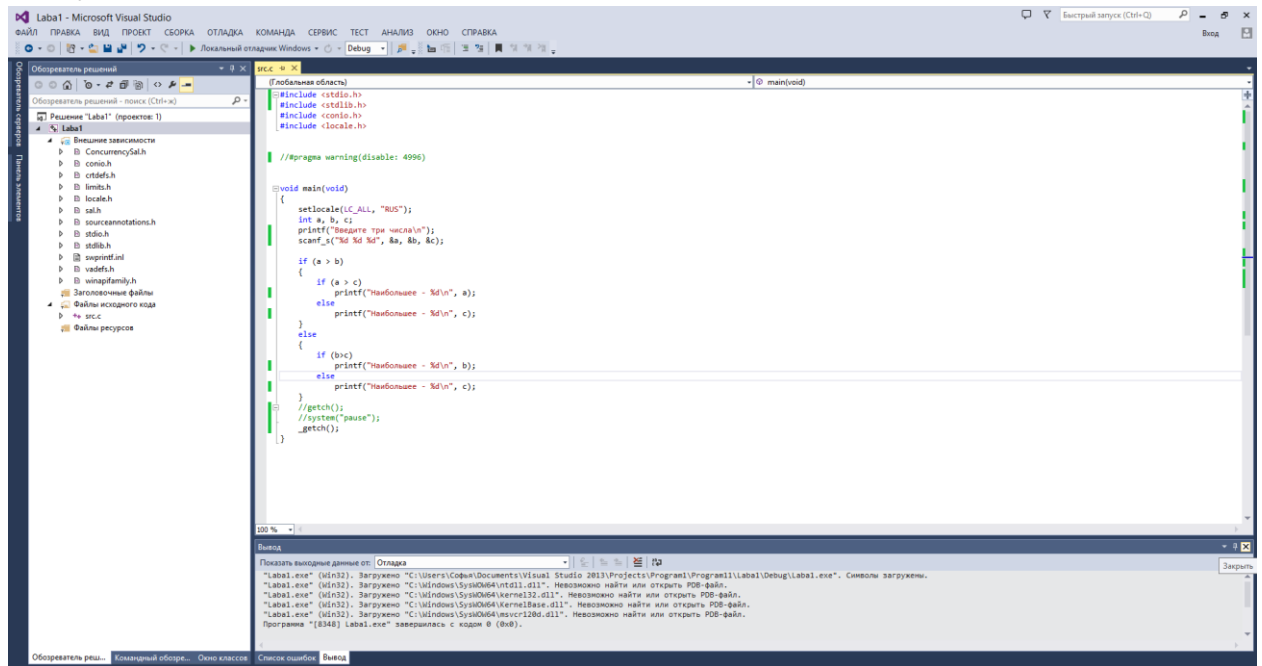
Для работы с русским языком:

`#include <locale.h>` - подключаем библиотеку

В теле функции используем:

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

1. Пункты меню



1.1. Пункт меню «Файл»:

- 1.1.1. Создать – пункт для создания нового проекта, файла и т.п.
- 1.1.2. Открыть – пункт для открытия уже существующих проектов, файла и т.п.
- 1.1.3. Добавить – позволяет создать/добавить существующий проект к текущему решению.
- 1.1.4. Закрыть – закрыть текущий файл.
- 1.1.5. Закрыть решение.
- 1.1.6. Сохранить имя_файла – сохранение (Ctrl+S).
- 1.1.7. Сохранить как имя_файла позволяет выбрать новое имя/расположение текущего файла.
- 1.1.8. Сохранить все – сохранение всего проекта.
- 1.1.9. Последние файлы – позволяет просмотреть недавно измененные файлы.
- 1.1.10. Последние проекты - позволяет просмотреть недавно измененные проекты.

1.2. Пункт меню «Правка»:

- 1.2.1. Отменить – отмена последнего действия (Ctrl+Z).
- 1.2.2. Вернуть – отмена отмены последнего действия (Ctrl+Y) (можно использовать, чтобы отменить последний Ctrl+Z).
- 1.2.3. Вырезать, копировать и вставить (Ctrl+X, Ctrl+C, Ctrl+V) – для редактирования кода.
- 1.2.4. Выделить все (Ctrl+A).
- 1.2.5. Поиск и замена (несколько пунктов, позволяющих быстро отредактировать какой-то фрагмент текста),
- 1.2.6. Перейти – переход к строке по номеру.
- 1.2.7. Перейти в – можно выбрать библиотеку/файл для перехода (по идее можно и функцию выбрать).
- 1.2.8. Вставить файл как текст – это замена тупым переносом файла целиком.
- 1.2.9. Структура – управление блоками текста, можно сворачивать/разворачивать блоки текста.

1.3. Пункт меню «Вид»:

- 1.3.1. Обозреватель решений – окно управления решениями.
- 1.3.2. Вывод – и так открыто по умолчанию. Можно смотреть логи сборки-отладки.
- 1.3.3. Классы (пригодится во втором семестре).
- 1.3.4. Окно свойств (пригодится во втором семестре).

1.4. Пункт меню «Проект»:

- 1.4.1. Добавить новый элемент – добавление нового файла к текущему проекту.
- 1.4.2. Добавить существующий элемент – добавление существующего файла к текущему проекту.
- 1.4.3. Добавить класс (пригодится во втором семестре).
- 1.4.4. Свойства конфигурации – настройки всей среды.

1.5. Пункт меню «Сборка»:

- 1.5.1. Собрать решение – полноценная сборка решения со всеми включенными проектами.
- 1.5.2. Построить решение – сборка текущего проекта в решении.

Примечание!

Команда "Собрать решение" компилирует все проекты в решении.
"Построить имя_проекта" компилирует текущий проект и его зависимости.
Если создан всего один проект в решении или текущий проект зависит от всех остальных, то тогда разницы между этими командами не будет.

- 1.5.3. Выполнить анализ кода в решении – классный инструмент для анализа кода. Как раз анализатор существующих проблем, находит семантические ошибки! (предупреждения, не обязательно ошибки). **Синтаксические – в окне ошибки!**

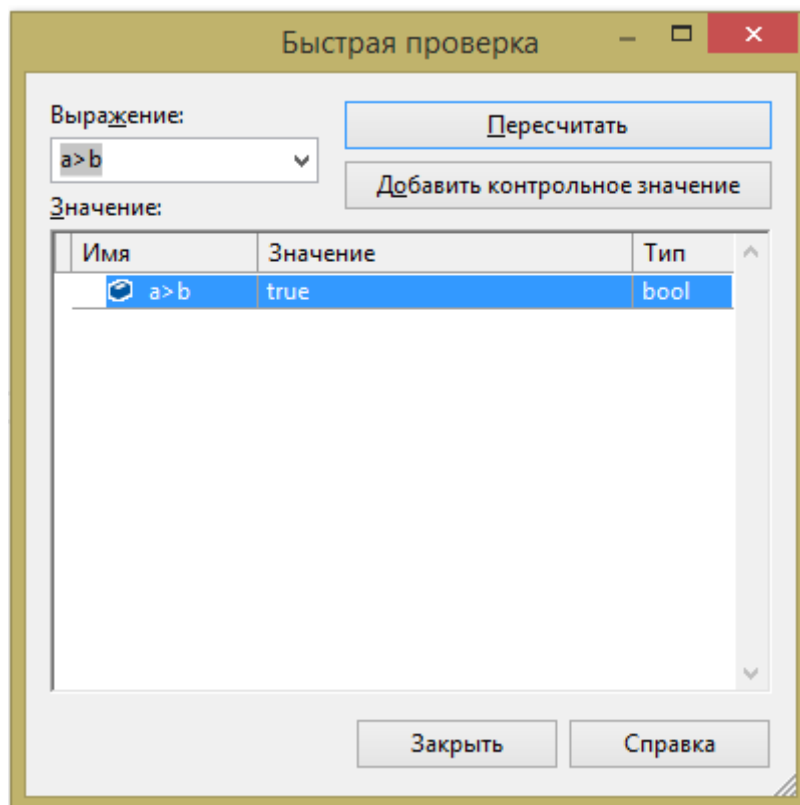
1.6. Пункт меню «Отладка»:

- 1.6.1. Окна – можно открыть окна «Точки останова», «Вывод» и «Интерпретация».
- 1.6.2. Начать отладку (F5). Запуск программы на исполнение.
- 1.6.3. Запуск без отладки – если по-простому – отладка работать не будет (Ctrl+F5):
- 1.6.4. Исключения (могут пригодиться во втором семестре).

В режиме отладки в собранный исполнительный бинарный файл добавляются специальные символы, позволяющие посмотреть, в какой функции программа завершилась аварийно. Отладочный бинарный файл позволяет пройти по стеку вызовов функций, выполнить каждую инструкцию процессора или C/C++-строчку по порядку. Поставить брекпойнт и т.д.

- 1.6.5. **Шаг с заходом (F11) – в подпрограммы.**
- 1.6.6. **Шаг с обходом (F10).**

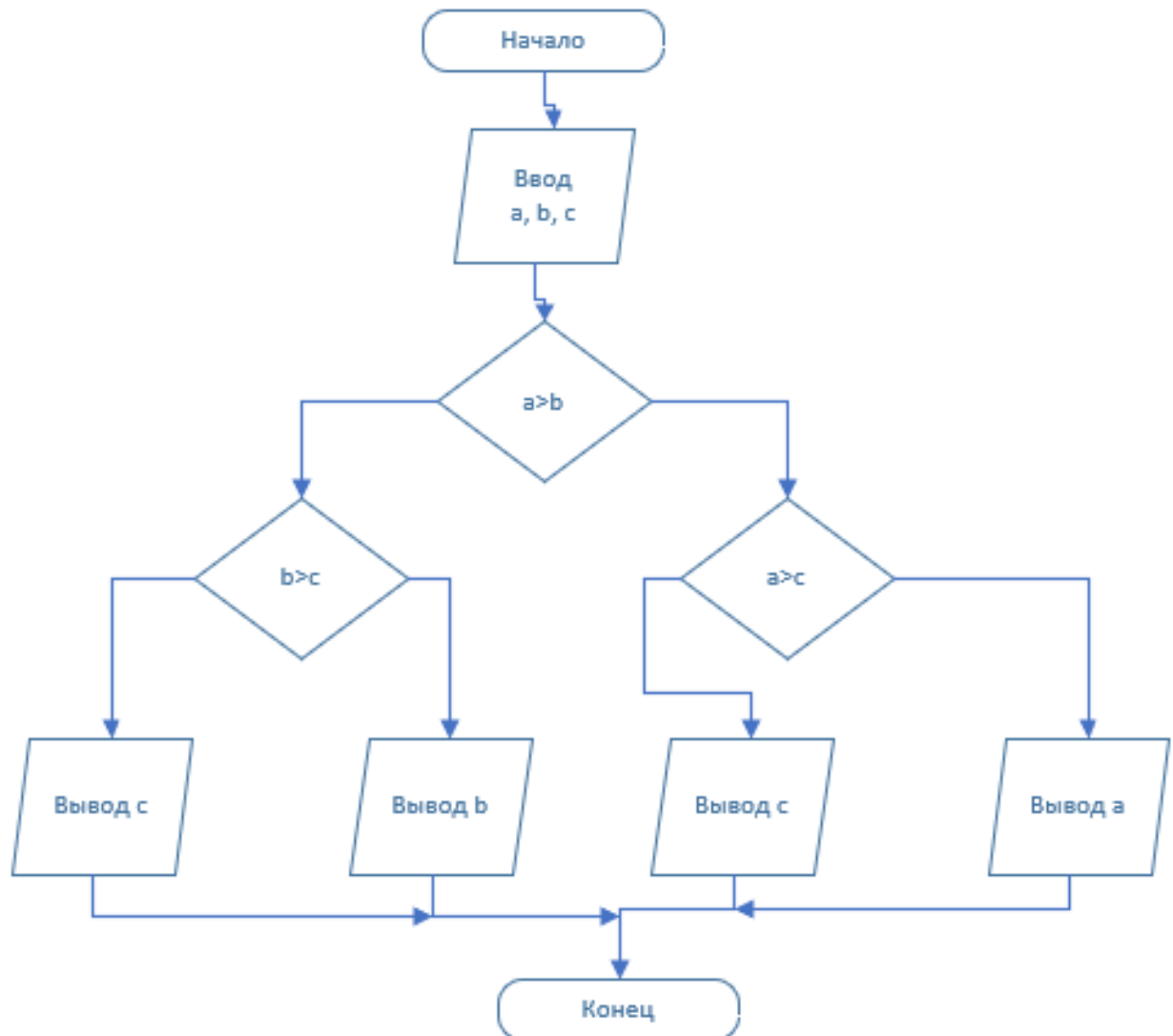
- 1.6.7. Шаг с выходом (Можно выйти из подпрограммы в родительскую).
(ТОЛЬКО ПРИ ИСПОЛНЕНИИ).
- 1.6.8. Быстрая проверка (ТОЛЬКО ПРИ ИСПОЛНЕНИИ)



- 1.6.9. Выключить все точки останова. Вкл. обратно этой же командой.
- 1.7. Пункт меню «Сервис»:**
- 1.7.1. Настройка
 - 1.7.2. Параметры

2. Решение задачи:

Задача: Даны три натуральных числа, попарно различные. Найти и вывести максимальное.



3. Пишем первую программу:

```
#include <stdio.h>
#include <locale.h>

#pragma warning(disable: 4996) //для отключения предупреждения об
использовании функции scanf и getch(). Но лучше использовать scanf_s и
_getch().

void main(void)
{
    setlocale(LC_ALL, "RUS");
    int a, b, c;
    printf("Введите три числа\n");
    scanf("%d %d %d", &a, &b, &c);

    if (a > b)
    {
        if (a > c)
            printf("Наибольшее - %d\n", a);
        else
            printf("Наибольшее - %d\n", c);
    }
    else
    {
        if (b > c)
            printf("Наибольшее - %d\n", b);
        else
            printf("Наибольшее - %d\n", c);
    }

    getch();
}
```

- сфера действия препроессора

include - подключить стандартную библиотеку, в данном случае библиотеку ввода- вывода

stdio.h - в этом файле находится описание функций ввода-вывода

main() - заголовок функции (не оператор, т.к. нет ; и вне { }. В () находятся аргументы (какие-либо переменные, которые функция может получать из других функций). У нас нет подпрограмм поэтому и нет переменных, которые передаем.

{- начало функции, } -конец функции.

int a, b, c; - описание переменных.

printf("Введите a,b,c \n"); - вывод сообщения пользователю, чтобы он с клавиатуры ввел числа
printf -библиотечная ф-ция вывода. В "" находится то, что необходимо вывести на экран.

\n - переход на следующую строку.

`scanf (" %d, %d, %d", &a,&b,&c);` - как и `printf`, печатает все, что в " " .

`%d` - целого типа

`%f` - вещественного типа

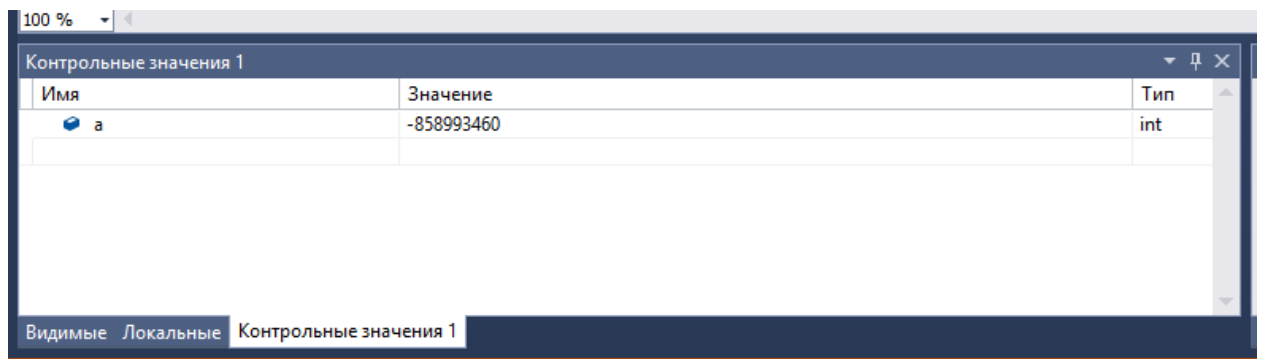
`%s` - печать строки символов.

`%c` - 1 символ

В `scanf` **&a, &b, &c** - занесение данных по адресу этой переменной.

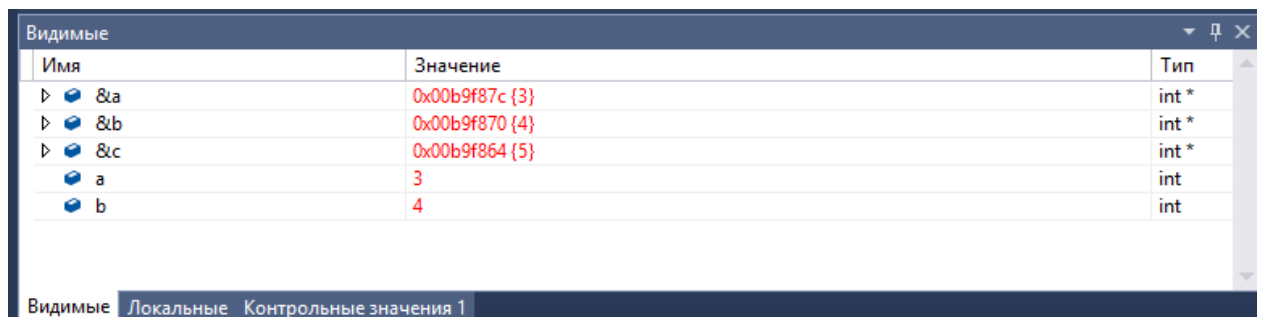
Запускаем отладку.

Внизу изучаем окно «Контрольные значения»:



Имя	Значение	Тип
a	-858993460	int

Как видим, есть еще два «Локальные» - для локальных переменных и «Видимые» - нет, не для глобальных, тут отображаются все производные от используемых переменных. Например, для нашей программы окно выглядит так:



Имя	Значение	Тип
&a	0x00b9f87c {3}	int *
&b	0x00b9f870 {4}	int *
&c	0x00b9f864 {5}	int *
a	3	int
b	4	int

Лабораторная работа №1.

Для выполнения лабораторной работы необходимо:

1. Получить индивидуальное задание.
2. Составить блок схему алгоритма решения задачи.
3. Написать программу на языке программирования Си в соответствии с алгоритмом, описанным блок схемой.
4. Отладить и протестировать программу.
5. Составить отчет о лабораторной работе.

Содержание отчета:

- Титульный лист.
- Условие задачи.
- Блок схема.
- Текст программы.
- Скриншоты, подтверждающие корректную работу программы с разными наборами входных данных (около 5 наборов).