**1.Основные принципы алгоритмизации и программирования. Алгоритмы и программы. Данные. Понятие типа данных. Логические основы алгоритмизации**.

Под **алгоритмом** понимали конечную последовательность точно сформулированных правил, которые позволяют решать те или иные классы задач.

**Свойства алгоритмов**.

Разделение выполнения решения задачи на отдельные операции (выполняемые исполнителем по определенным командам) — важное свойство алгоритмов, называемое **дискретностью**.

Алгоритм, составленный для конкретного исполнителя, должен включать только те команды, которые входят в его систему команд. Это свойство алгоритма называется **понятностью**.

Еще одно важное требование, предъявляемое к алгоритмам, — **результативность (или конечность)** алгоритма. Оно означает, что исполнение алгоритма должно закончиться за конечное число шагов.

**Формы записи алгоритмов**

· записан на естественном языке (примеры записи алгоритма на естественном языке приведены при определении понятия алгоритма);

· изображен в виде блок-схемы;

· записан на алгоритмическом языке

**Базовые структуры алгоритмов** — это определенный набор блоков и стандартных способов их соединения для выполнения типичных последовательностей действий. К основным структурам относятся следующие: **линейные** , **разветвляющиеся** , **циклические** .

**Линейными** называются алгоритмы, в которых действия осуществляются последовательно друг за другом.

**Разветвляющимся** называется алгоритм, в котором действие выполняется по одной из возможных ветвей решения задачи, в зависимости от выполнения условий.

**Циклическим** называется алгоритм, в котором некоторая часть операций (тело цикла — последовательность команд) выполняется многократно.

**Данные** — это любая информация, представленная в формализованном виде и пригодная для обработки алгоритмом.

**Данные** делятся на **переменные** и **константы**.

**Переменные** — это такие данные, значения которых могут изменяться в процессе выполнения алгоритма.

**Константы** — это данные, значения которых не меняются в процессе выполнения алгоритма.

Типы данных принято делить на **простые (базовые)** и **структурированные**.

К основным базовым типам относятся:

· **целый (INTEGER)** — определяет подмножество допустимых значений из множества целых чисел;

· **вещественный (REAL,FLOAT)** — определяет подмножество допустимых значений из множества вещественных чисел;

· **логический (BOOLEAN)** — множество допустимых значений — истина и ложь;

· **символьный (CHAR)** — цифры, буквы, знаки препинания и пр.

**Структурированные** типы описывают наборы однотипных или разнотипных данных, с которыми алгоритм должен работать как с одной именованной переменной.

Наиболее широко известная структура данных — **Массив**. **Массив -**представляет собой упорядоченную структуру однотипных данных, которые называются элементами массива.

**Логические основы алгоритмизации**.

Если высказывание С истинно, то пишут С = 1 (С = t, true), а если оно ложно, то С = 0 (С = f, false).

Для образования новых высказываний наиболее часто используются логические операции, выражаемые словами **«не», «и», «или».**

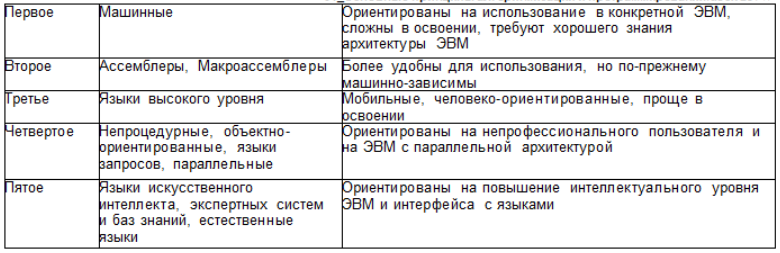
**Конъюнкция (логическое умножение).** Соединение двух (или нескольких) высказываний в одно с помощью союза И (OR) называется операцией, логического умножения, или конъюнкцией.

Объединение двух (или нескольких) высказываний с помощью союза ИЛИ (OR) называется операцией **логического сложения, или дизъюнкцией**. Эту операцию обозначают знаками « | , v» или знаком сложения «+». Сложное высказывание A v В истинно, если истинно хотя бы одно из входящих в него высказываний.

Присоединение частицы НЕ (NOT) к данному высказыванию называется **операцией отрицания (инверсии)**. Она обозначается Ă (или ¬A)и читается не А. Если высказывание А истинно, то В ложно, и наоборот.

# 

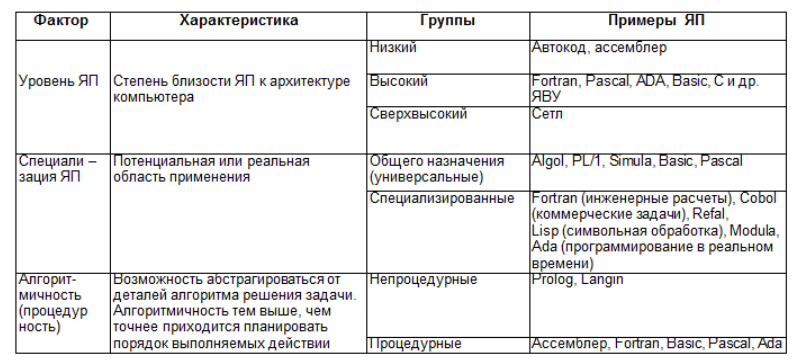
**2.Основные принципы алгоритмизации и программирования. Языки программирования: эволюция, классификация. Системы программирования. Файлы данных.**

****

Вычислительная машина (система), независимо от типа и поколения, состоит из двух основных типов устройств:

· центральное устройство (ЦУ), включающее в себя центральный процессор (ЦП) и оперативную память (ОП);

· периферийные (внешние) устройства (ВУ).

**Классификация ЯП**. Изучение ЯП часто начинают с их классификации. Определяющие факторы классификации обычно жестко не фиксируются..

**Системы программирования** ­­­– это комплекс инструментальных программных средств, предназначенных для работы с программами на одном из языков программирования. Системы программирования представляют сервисные возможности программистам для разработки их собственных компьютерных программ.

**Файлы данных.**

Ввод-вывод данных в языках программирования осуществляется путем взаимодействия программы с **внешними файлами**. **Внешний файл** — это поименованный файл на диске или устройство ввода-вывода (например, клавиатура или дисплей). Во внешних файлах сохраняются результаты работы программы и располагаются данные, служащие источником информации, необходимой для ее функционирования.**Файл** можно определить как логически непрерывный именованный набор данных на внешнем носителе.

**Могут быть рассмотрены следующие типы файлов:**

по типу записей:

· файлы с записями фиксированной длины;

· файлы с записями переменной или неопределенной длины;

· файлы, образующие байтовый или битовый поток;

по способу выборки информации:

· файлы последовательного доступа;

· файлы прямого доступа.

**Файлы последовательного доступа**. Рассмотрим вначале файлы последовательного доступа. Для таких файлов характерны операции последовательного чтения и записи в конец файла

**Файлы прямого доступа**. Для файлов прямого доступа характерны операции чтения и записи по произвольному адресу.

# 

**3.Основные принципы алгоритмизации и программирования. Объектно-ориентированный подход к программированию. Разработка программного обеспечения (ПО).**

**Одной из первых и наиболее широко применяемых технологий программирования стало структурное программирование.**

**Структурный подход базируется на двух основополагающих принципах:**

· использование процедурного стиля программирования;

· последовательная декомпозиция алгоритма решения задачи сверху вниз.

**Объектно-ориентированное программирование** родилось и получило широкое распространение именно благодаря попыткам разрешения следующих проблем, возникавших в процессе проектирования и разработки программных комплексов.

1. Развитие языков и методов программирования не успевало за все более растущими потребностями в прикладных программах. Единственным реальным способом снизить временные затраты на разработку был метод многократного использования разработанного программного обеспечения, т.е. проектирование новой программной системы на базе разработанных и отлаженных ранее модулей, которые выступают в роли своеобразных «кирпичиков», ложащихся в Фундамент новой разработки.

2. Ускорение разработки программного обеспечения требовало решения проблемы упрощения их сопровождения и модификации.

3. Не все задачи поддаются алгоритмическому описанию по требованиям структурного программирования, поэтому в целях упрощения процесса проектирования необходимо было решить проблему приближения структуры программы к структуре решаемой задачи.

Существуют некоторые общие принципы, которые следует использовать при разработке ПО.

**Частотный принцип.** Основан на выделении в алгоритмах и в обрабатываемых структурах действий и данных по частоте использования. К данным, которым происходит частое обращение, обеспечивается наиболее быстрый доступ, а подобные операции стараются сделать более короткими.

**Принцип модульности**. Под модулем в общем случае понимают функциональный элемент рассматриваемой системы, имеющий оформление, законченное и выполненное в пределах требований системы, и средства сопряжения с подобными элементами или элементами более высокого уровня данной или другой системы.

**Принцип функциональной избирательности**. Этот принцип является логическим продолжением частотного и модульного принципов и, используется при проектировании ПО, объем которого существенно превосходит имеющийся объем оперативной памяти.

**Принцип генерируемости**. Данный принцип определяет такой способ исходного представления ПО, который бы позволял осуществлять настройку на конкретную конфигурацию технических средств, круг решаемых проблем, условия работы пользователя.

**Принцип функциональной избыточности**. Этот принцип учитывает возможность проведения одной и той же работы (функции) различными средствами. Особенно важен учет этого принципа при разработке пользовательского интерфейса для выдачи данных из-за психологических различий в восприятии информации.

**Принцип «по умолчанию».** Применяется для облегчения организации связей с системой, как на стадии генерации, так и при работе с уже готовым ПО. Принцип основан на хранении в системе некоторых базовых описаний структур, модулей, конфигураций оборудования и данных, определяющих условия работы с ПО. Эту информацию ПО использует в качестве заданной, если пользователь забудет или сознательно не конкретизирует ее. В данном случае ПО само установит соответствующие значения.

**4.Пакет компиляторов Visual C++. Рекомендуемое оборудование. Минимальные требования к аппаратному и программному обеспечению. Рекомендуемое аппаратное и программное обеспечение.**

**Минимальные требования к аппаратному и программному обеспечению**

Стандартный пакет компиляторов Microsoft Visual C++ работает на обширном семействе компьютеров с процессорами Intel. Существуют специальные версии Visual C++ для компьютеров Macintosh, MIPS и DEC Alpha AXP.

Ниже перечислены минимальные требования Microsoft к оборудованию и программному обеспечению для работы 16- и 32-разрядной версии Microsoft Visual C++:

· ПК с процессором 80386

· 16MB ОЗУ (минимум)

· 10МВ дискового пространства для минимальной установки

· Microsoft Windows 95 или Windows NT для разработки приложений Win32

· Дисплей VGA

· Дисковод для дискет высокой плотности

· Привод CD-ROM (для электронной документации)

· Мышь Microsoft

**Рекомендуемое аппаратное и программное обеспечение.**

Для оптимизации цикла разработки программ на С и C++ мы рекомендуем следующую конфигурацию системы:

· ПК с процессором Pentium, тактовая частота 90 МГц (или выше)

· 20МВ ОЗУ

· 100МВ свободного дискового пространства

· Microsoft Windows 95 или Windows NT для разработки приложений Win32

· Дисплей Super VGA

· Дисковод для 3.5-дюймовых дискет высокой плотности

· Привод CD-ROM (для электронной документации)

· Мышь Microsoft

Среда Visual Studio 2017 поддерживается на следующих операционных системах:

* [Windows 7 с Service Pack 1](https://info-comp.ru/vseowindowsst/509-installing-service-pack-1-windows-7.html);
* Windows 8.1;
* Windows 10;
* Windows Server 2012 R2;
* [Windows Server 2016](https://info-comp.ru/sisadminst/543-install-and-review-of-windows-server-2016.html).

Минимальные требования к оборудованию:

* Процессор с частотой не ниже 1,8 ГГц. Рекомендуется использовать как минимум двухъядерный процессор;
* 2 ГБ оперативной памяти, рекомендуется 4 ГБ;
* Свободного места на жестком диске от 1 ГБ до 40 ГБ, в зависимости от установленных компонентов;
* Видеоадаптер с минимальным разрешением 1280 на 720 пикселей.

**5. Пакет компиляторов Visual C++. Выбор правильных параметров установки. Какую конфигурацию выбрать?. Обычная установка под Windows. Каталоги.**

Установите Visual Studio 2017

**Шаг 1 - Убедитесь, что ваш компьютер готов для Visual Studio**

Прежде чем приступать к установке Visual Studio:

1. Проверьте [системные требования](https://www.visualstudio.com/productinfo/vs2017-system-requirements-vs) . Эти требования помогут вам узнать, поддерживает ли ваш компьютер Visual Studio 2017.
2. Примените последние обновления Windows. Эти обновления гарантируют, что на вашем компьютере установлены последние обновления безопасности и необходимые системные компоненты для Visual Studio.
3. Перезагружать. Перезагрузка гарантирует, что любые ожидающие установки или обновления не будут препятствовать установке Visual Studio.
4. Освободите место. Удалите ненужные файлы и приложения из вашего% SystemDrive%, например, запустив приложение «Очистка диска».

Вопросы о работе предыдущих версий Visual Studio рядом с Visual Studio 2017 см. В сведениях о [совместимости Visual Studio](https://www.visualstudio.com/productinfo/vs2017-compatibility-vs#compatibility-with-previous-releases) .

**Шаг 2 - Загрузить Visual Studio**

Затем загрузите файл начальной загрузки Visual Studio. Для этого щелкните следующую кнопку, выберите версию Visual Studio 2017, которую вы хотите, нажмите « **Сохранить»** , а затем « **Открыть папку»** .

**Шаг 3 - Установите установщик Visual Studio**

Затем запустите файл bootstrapper, чтобы установить установщик Visual Studio. Этот новый легкий установщик включает все необходимое для установки и настройки Visual Studio 2017.

1. В папке « **Загрузка** » дважды щелкните загрузочный файл, соответствующий или похожий на один из следующих файлов:
   * **vs\_enterprise.exe** для Visual Studio Enterprise
   * **vs\_professional.exe** для Visual Studio Professional
   * **vs\_community.exe** для сообщества Visual Studio

Если вы получили уведомление об управлении учетными записями пользователей, нажмите « **Да»** .

1. Мы попросим вас подтвердить [условия лицензии](https://www.visualstudio.com/license-terms/) Microsoft и [заявление о конфиденциальности](https://go.microsoft.com/fwlink/?LinkID=824704) Microsoft . Нажмите « **Продолжить»** .

**Шаг 4 - Выбор рабочих нагрузок**

После установки установщика вы можете использовать его для настройки своей установки, выбрав нужные вам наборы функций или рабочие нагрузки. Вот как.

1. Найдите требуемую рабочую нагрузку на экране « **Установка Visual Studio»** .

Например, выберите рабочую нагрузку «.NET desktop development». Он поставляется с основным редактором ядра, который включает в себя базовую поддержку редактирования кода на более чем 20 языках, возможность открывать и редактировать код из любой папки без необходимости проекта и интегрированного управления исходным кодом.

1. После того, как вы выберете требуемую рабочую нагрузку (ы), нажмите « **Установить»** .

Затем появятся экраны состояния, показывающие ход установки Visual Studio.

1. После установки новых рабочих нагрузок и компонентов нажмите « **Запустить»** .

**Шаг 5 - Выберите отдельные компоненты (необязательно)**

Если вы не хотите использовать функцию Workloads для настройки вашей Visual Studio, вы можете сделать это, установив вместо этого отдельные компоненты. Чтобы выбрать отдельные компоненты, выберите параметр « **Отдельные компоненты»** в установщике Visual Studio, выберите нужное, а затем следуйте инструкциям.

**Шаг 6 - Установите языковые пакеты (необязательно)**

По умолчанию программа установки пытается совместить язык операционной системы, когда она запускается в первый раз. Чтобы установить Visual Studio 2017 на выбранном вами языке, выберите параметр « **Языковые пакеты»** из установщика Visual Studio и следуйте инструкциям.

Измените язык установщика из командной строки

Другой способ, с помощью которого вы можете изменить язык по умолчанию, - запустить установщик из командной строки. Например, вы можете заставить программу установки работать на английском языке, используя следующую команду: vs\_installer.exe --locale en-US. Установщик помнит этот параметр, когда он будет запущен в следующий раз. Установщик поддерживает следующие языковые токены: zh-cn, zh-tw, cs-cz, en-us, es-es, fr-fr, de-de, it-it, ja-jp, ko-kr, pl- pl, pt-br, ru-ru и tr-tr.

**Шаг 7 - Начните разработку**

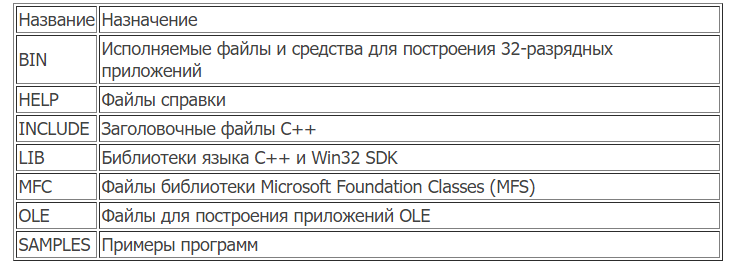
1. После завершения установки Visual Studio нажмите кнопку « **Запустить»** , чтобы [начать работу с Visual Studio](https://docs.microsoft.com/en-us/visualstudio/ide/get-started-developing-with-visual-studio) .
2. Нажмите « **Файл»** , а затем « **Новый проект»** .
3. Выберите тип проекта.   
   Например, чтобы [создать приложение C ++](https://docs.microsoft.com/en-us/visualstudio/ide/getting-started-with-cpp-in-visual-studio) , нажмите « **Установлено»** , разверните **Visual C ++** и выберите тип проекта C ++, который вы хотите создать.   
   Чтобы [создать приложение C #](https://docs.microsoft.com/en-us/visualstudio/ide/walkthrough-create-a-simple-application-with-visual-csharp-or-visual-basic) , нажмите « **Установить»** , разверните **Visual C #** и выберите тип проекта C #, который вы хотите создать.

*Получать поддержку:*

Иногда все может пойти не так. Если ваша установка Visual Studio не удалась, см. Страницу « [Устранение неполадок с Visual Studio 2017» и «Проблемы с обновлением»](https://docs.microsoft.com/en-us/visualstudio/install/troubleshooting-installation-issues) . Если ни один из шагов по устранению неполадок не помогает, вы можете связаться с нами через чат для помощи по установке (только на английском языке). Подробнее см. На [странице поддержки Visual Studio](https://www.visualstudio.com/vs/support/#talktous) .

**Каталоги.**

В таблице показана типичная группа подкаталогов для Visual C++ 4.0, установленного в каталог MSdev.



Документация по Visual C++ состоит из Quick Reference (Быстрой справки) и Books Online (Электронных книг). Быстрая справка позволит вам быстро отыскать информацию в процессе программирования. Электронные книги представляют собой полный набор документации по Visual C++ в компьютерном формате.

В документации содержатся, в частности, следующие разделы:

· Как пользоваться Электронными книгами

· Различные Руководства пользователя

· Microsoft Foundation Classes (MFC)

· Программирование с использованием библиотеки MFC

· Справочник по библиотеке MFC

· Примеры по библиотеке MFC

· Технические указания по библиотеке MFC/C++

**6.Пакет компиляторов Visual C++. Система разработки. Новый встроенный отладчик. Новые встроенные редакторы ресурсов. Дополнительное средство TestContainer.**

**Система разработки.**

32-разрядный компилятор Microsoft Visual C++ для Windows NT и Windows 95 вобрал в себя новейшие средства разработки для Windows, тесно связанные друг с другом и снабженные наглядным интерфейсом. Microsoft использовала всю мощь своего отладчика CodeView непосредственно в среде Visual C++. Отладчик доступен из меню Build (сборка). Встроенный отладчик позволит вам выполнять программу в пошаговом режиме, считывать и изменять содержимое переменных, и даже двигаться назад по коду. Доступ к этим редакторам осуществляется из меню Insert (вставить). Редакторы ресурсов позволяют проектировать и создавать ресурсы Windows: растровые изображения, курсоры, значки, меню и окна диалога.

**Test Container** (тестовый контейнер) — это спроектированное Microsoft приложение, которое позволяет быстро тестировать ваши управляющие элементы. Можно изменять свойства и возможности элемента, находящегося в тестовом контейнере.

Отдельные инструменты расположены вне интегрированной среды разработки (Integrated Development Environment, IDE). Некоторые из них, как например Spy++ и MFC Tracer, доступны и внутри интегрированной среды, и вне ее.

**7.Пакет компиляторов Visual C++. Инструменты, не вошедшие в интегрированную среду. ProcessViewer (PView). WinDiff.**

**Инструменты, не вошедшие в интегрированную среду**

**Process Viewer** (наблюдение за процессами) позволяет быстро устанавливать параметры, необходимые для отслеживания выполняемых процессов, потоков и квантования процессорного времени. Чтобы запустить Process Viewer, просто дважды щелкните на значке PView в группе Visual C++.

**WinDiff.**

Утилита WinDiff также находится в группе Visual C++. Этот средство позволит вам в графическом виде сравнивать и изменять два файла или два каталога. Все возможности WinDiff работают очень сходно с соответствующими командами Windows 95 Explorer или Windows NT File Manager.

**8.Пакет компиляторов Visual C++. Важные возможности компилятора. P-код. Предварительно откомпилированные заголовки и типы. Библиотека MicrosoftFoundationClass. Встраивание функций.**

**Важные возможности компилятора**

Пакет компилятора Visual C++ включает множество усовершенствований, новых возможностей и дополнений. Следующие разделы представят вам эти улучшения и кратко пояснят их применение.

**P-код** (сокращение от packed code — упакованный код) нацелен на оптимизацию размера и быстродействия кода. Р-код может существенно уменьшить размер программы и повысить скорость ее выполнения на величину до 60 процентов. Более того, все это достигается простым включением определенного режима компиляции. Это означает, что любая программа на С или C++ может компилироваться как обычным способом, так и с применением р-кода.

Visual C++ помещает родовые типы, прототипы функций, внешние ссылки и объявления функций-членов класса в специальные файлы, называемые заголовочными. Эти файлы содержат много важных определений, необходимых многочисленным исходным файлам, которые связываются воедино для создания исполняемой программы. Части заголовочных файлов, как правило, компилируются заново для каждого модуля, в который он включается. К сожалению, необходимость повторной компиляции участков текста приводит к снижению общей производительности компилятора. Visual C++ ускоряет процесс компиляции, позволяя вам заранее откомпилировать ваши заголовочные файлы. Хотя **принцип предварительной компиляции** не нов, Microsoft использует действительно новый подход. Предварительная компиляция сохраняет на определенном этапе состояние компилируемой программы и вводит соотношение между файлами с текстом программы и заранее откомпилированным заголовком. Можно создать и больше одного откомпилированного заголовочного файла на каждый из файлов с текстом программы.

Библиотека **MFC** предоставляет классы для управления объектами Windows и обладает рядом классов общего назначения, которые могут быть использованы и в приложениях MS-DOS, и Windows. Например, есть классы для управления файлами, строками, временем, памятью и обработки исключительных ситуаций.

Компилятор Microsoft Visual C++ обеспечивает полную поддержку **встраиваемых функций**. Это означает, что функция, содержащая комбинацию команд любого типа, может быть встроена в программу в месте ее вызова. Многие распространенные компиляторы C++ не допускают встраивание определенных операторов или выражений, — например, встраивание не применяется к функциям, содержащим оператор switch, while или for.

Кроме обычных функций в C++, о которых вы уже знаете, есть еще встроенные функции. Встроенные функции не так значимы, но желательно в них разбираться. Основная идея в том, чтобы ускорить программу ценой занимаемого места. Встроенные функции во многом похожи на заполнитель. После того как вы определите встроенную функцию с помощью ключевого слова inline, всякий раз когда вы будете вызывать эту функцию, компилятор будет заменять вызов функции фактическим кодом из функции.

Как это делает программу быстрее? Легко, вызовы функций занимают больше времени, чем написание всего кода без функции. Просмотр вашей программы и замена функции, которую вы использовали 100 раз с кодом из функции, займет очень много времени. Конечно, используя встроенные функции для замены обычных вызовов функций, вы также значительно увеличите размер вашей программы.

Использовать ключевое слово inline легко, просто поставьте его перед именем функции. Затем, используйте её как обычную функцию.

**9.Пакет компиляторов Visual C++. Параметры компилятора. General. Debug. CustomBuild.**

Пакет Microsoft Visual C++ включает в себя средства для построения программ для Windows. Кроме этого, ваш код может быть адаптирован для таких аппаратных платформ, как Apple Macintosh и машин с RISC-процессорами. В пакете C++ имеются все необходимые заголовочные файлы, библиотеки, редакторы окон диалога и ресурсов, необходимые для создания действительно надежного приложения Windows. Microsoft также включила непосредственно в среду разработки редакторы ресурсов для пиктограмм, растровых изображений, курсоров, меню и окон диалога.

**Параметры компилятора**

Рассматриваемые ниже параметры позволяют оптимизировать скорость, размер исполняемого модуля или время компиляции и сборки. Если вы не наблюдаете заметного скачка производительности, то, возможно, ваше приложение содержит недостаточно кода. Все параметры устанавливаются путем выбора пункта Settings (параметры) из меню Build (сборка).

Закладка **General** (общие параметры) позволяет указать, следует или нет использовать библиотеку **Microsoft Foundation Class**. Можно также задать каталоги, куда будут помещены промежуточные и окончательные результаты компиляции.

Закладка **Debug** (отладка) позволяет указать местоположение исполняемого файла и рабочий каталог, дополнительные параметры для программы, а также путь и имя файла для дистанционной отладки.

При помощи закладки **Custom Build** (дополнительные средства) вы можете задать дополнительные инструменты для построения программы. Сюда входят, например, программы, обрабатывающие выходной файл конфигурации проекта.

**10.Пакет компиляторов Visual C++. C/С++. C++ Language. CodeGeneration. Customization. ListingFiles. Optimizations. PrecompiledHeaders. Preprocessor.**

**C/С++**

Закладка C/C++ позволяет выбрать одну из следующих категорий:

· General

· С++ Language

· Code Generation

· Customization

· Listing Files

· Optimizations

· Precompiled Headers

· Preprocessor

**C++ Language.**

Категория **Code Generation** (генерация кода) позволяет задать целевой микропроцессор (от 80386 до Pentium), выбрать соглашение о вызовах, указать стандартную библиотеку и задать метод выравнивания элементов структуры.

Категория **Customization** (адаптация) позволяет включить или отключить следующие возможности:

· расширения языка

· компоновка на уровне функций

· идентичные строки

· минимальная перекомпиляция

· инкрементная компиляция

· отмена заставки и информационных сообщений

Категория **Listing Files** (файлы листинга) позволяет включить генерацию информации для браузера. Кроме того, можно указать местоположение файла для браузера. Можно разрешить включение локальных переменных в информацию браузера. Дополнительно можно задать типы файлов. Перечислены параметры проекта.

Категория **Optimizations** (параметры оптимизации) позволяет установить различные варианты оптимизации, например, по скорости или по размеру.

Категория **Precompiled Headers** (предварительно откомпилированные заголовки) позволяет использовать заранее откомпилированные заголовочные файлы. Это файлы с расширением РСН. Категория **Preprocessor** (препроцессор) позволяет задать макроопределения для препроцессора. Можно также указать дополнительные каталоги и отменить стандартные пути.

**11.Пакет компиляторов Visual C++. Link. General. Customization. Debug. Input. Output.**

**C++ Language**

Закладка **Link** (компоновка) позволяет выбрать одну из следующих категорий: - General -Customization - Debug - Input - Output

Категория **General** (общие параметры) позволяет установить порог выдачи предупреждающих сообщений, указать отладочную информацию, установить оптимизацию компиляции и перечислить дополнительные параметры проекта.

Категория **Customization** (адаптация) позволяет включить или отключить следующие возможности:

· расширения языка

· компоновка на уровне функций

· идентичные строки

· минимальная перекомпиляция

· инкрементная компиляция

· отмена заставки и информационных сообщений

Категория **Debug** (отладка) позволяет задать различные форматы для создания карты компоновки и отладочной информации.

Категория **Input** (ввод) позволяет указать объектные модули и библиотеки. Кроме того, можно задать имена файла перекрестных ссылок и файла, где находится DOS-заголовок программы.

Категория **Output** (вывод) позволяет установить базовый адрес, точку входа, размер стека и информацию о версии проекта.

**12.Пакет компиляторов Visual C++. Resources. OLE Types. BrowseInfo.**

**C++ Language**

Закладка **Resources** (ресурсы) позволяет задать имя файла ресурсов (обычно он имеет расширение .RES). Можно дополнительно указать язык ресурсов, каталоги для ресурсов и задать препроцессорные определения.

Закладка **OLE Types** (типы OLE) позволяет задать имя выходного файла, имя выходного заголовочного файла, определения для препроцессора и заставку.

Закладка **Browse Info** (информация браузера) дает возможность указать имя файла с информацией браузера. Кроме того, можно включить возможность генерации информации браузера и заставки.

**13.Меню File (файл). New... Open... Close. Open Workspace. Close Workspace. Save. Save As...**

В меню File среды Visual C++ собраны стандартные средства работы с файлами, встречающиеся во многих приложениях Windows.

**New...**

Пункт **New...** (создать) открывает новое окно редактирования файла. Обычно с этого начинается создание программы. Среда автоматически называет и нумерует открытые вами окна. Нумерация начинается с 1, поэтому заголовок первого окна всегда будет хxx/, второго окна — ххх2, и так Далее, ххх соответствует типу файла, с которым вы работаете (текст, проект, ресурс, рисунок, код, значок или курсор).

**Open...**

В отличие от **New...,** который открывает окно редактирования не существовавшего ранее файла, пункт **Open...** (открыть) вызывает появление окна диалога с запросом информации о файле, который уже сохранен на диске. Это стандартное окно диалога **Open File** (открыть файл), в котором показаны текущее устройство, каталог и шаблон поиска файлов, и предлагается ввести нужные параметры.

**Close.**

Команда **Close** (закрыть) используется, чтобы закрыть открытый файл. Если у вас открыто несколько файлов, будет закрыто активное (текущее) окно. Вы можете отличить активное окно от неактивного, посмотрев на рамку окна. Активное (текущее) окно имеет фокус ввода и изображается цветом, установленным в системе для отображения активных окон. Обычно для активных окон установлены выделенный цветом заголовок и темная рамка. Неактивные окна имеют обычно серые заголовок и рамку.

**Open Workspace.**

Команда **Open Workspace** (открыть рабочее пространство) используется для активизации ранее сохраненного рабочего пространства. Рабочие пространства соответствуют приложениям, которые вы можете создавать. Проект состоит из одного набора исходных файлов и набора из одной или более конфигураций. Каждая конфигурация проекта вместе с набором файлов однозначно определяют двоичный файл, который в результате будет генерироваться.

**Close Workspace.**

Команда **Close Workspace** (закрыть рабочее пространство) закрывает активное рабочее пространство. Это позволяет открыть другое рабочее пространство и перейти к работе над другим приложением.

**Save.**

Команда **Save** (сохранить) записывает содержимое активного окна в соответствующий файл. Вы можете определить, существует ли такой файл, просто взглянув на заголовок окна. Если вы увидите автоматически созданный заголовок (вроде ххх1), значит, содержимое окна еще ни разу не было сохранено, и ему не назначено имени файла. Вы также можете сохранить файлы, нажав соответствующую кнопку панели инструментов.

**Save As...**

Команда **Save As...** (сохранить как) позволяет создать копию содержимого активного окна под другим именем. Например, вы закончили работу над проектом. У вас есть работающая программа, но вы все же хотите попробовать внести пару изменений. Ради собственного спокойствия вы не хотите трогать работающую версию. Воспользовавшись командой Save As, вы можете сохранить файл под другим именем, и затем работать с его копией. Если новая версия работать не будет, можно вернуться к первоначальной версии файла.

**14.Меню File (файл). Save All. Find in Files... Page Setup... Print... Список последних проектов. Exit.**

В меню File среды Visual C++ собраны стандартные средства работы с файлами, встречающиеся во многих приложениях Windows.

**Save All.**

Команда **Save All** (сохранить все) записывает в соответствующие файл, содержимое всех открытых окон. Если вы никогда не писали приложении Windows на С или C++, вас, возможно, ошеломит общее количество файлов, требуемых для создания работающей программы. Недостаток команды **Save** заключается в том, что она сохранит лишь содержимое активного окна.

**Find in Files...**

Команда **Find in Files** (поиск в файлах) позволяет найти последовательность символов в одном или нескольких файлах. Файлы, в которых следует искать, задаются указанием типа файла и каталога, где они находятся. Для задания искомой последовательности вы можете пользоваться регулярными выражениями. Результаты поиска отображаются в окне Output (вывод). Когда поиск закончен, можно открыть нужный файл, дважды щелкнув на его имени в окне Output.

**Page Setup...**

Наиболее часто команда **Page Setup** (настройка параметров страницы) используется при оформлении распечаток текстов программ. Эта команда позволяет выбрать для каждой страницы верхний и нижний колонтитул, а также установить размеры полей.

**Print...**

Чтобы получить на бумаге копию содержимого активного окна, достаточно выполнить команду **Print** (печать). Окно диалога **Print** предоставляет вам несколько возможностей. Во-первых, вы можете решить, печатать ли все содержимое окна, или только выделенный текст. Если в окне есть выделенный текст, кнопка опции **Print Range Selection** (печатать только выделенный) будет изображена обычным цветом (не серым), указывая на то, что данная возможность доступна.

**Список последних проектов.**

**Список последних использовавшихся проектов** расположен в меню **File** непосредственно под хронологическим списком файлов. Список проектов схож со списком файлов, но содержит только имена проектов. Чтобы открыть любой файл из этих списков, щелкните левой кнопкой мыши на нужном имени.

**Exit.**

Команда **Exit** (выход) позволяет завершить работу среды Visual C++. Не волнуйтесь, если вы забыли сохранить какие-либо файлы; среда автоматически выдаст предупреждение и даст возможность сохранить все измененные файлы.

**15.Меню Edit. Undo. Redo. Cut. Сору. Paste. Delete. Select All.**

Команды меню Edit (правка) позволяют быстро находить и исправлять текст в активном окне примерно так же, как это делается любым из распространенных текстовых процессоров.

**Undo.**

Команда **Undo** (отменить) позволяет отменить последнее изменение, которое было внесено в процессе редактирования. Вы можете также выполнить эту команду с помощью панели инструментов. На кнопке **Undo** изображена стрелочка влево; эта кнопка расположена седьмой слева.

**Redo.**

Команда **Redo** (восстановить) позволяет отменить действие последней команды **Undo**. Пользуйтесь этой командой, чтобы вернуть те изменения, которые вы по ошибке отменили. Команда **Redo** доступна также посредством панели инструментов. Это восьмая кнопка слева.

**Cut.**

Команда **Cut** (вырезать) переносит в буфер обмена текст, выделенный в активном окне. При этом выделенный текст будет удален из окна. Для того чтобы выделить текст, поместите указатель мыши на первый из символов выделяемого текста и, удерживая левую кнопку нажатой, переместите мышь вправо/вниз до конца выделяемого участка. После этого текст, который вы выделили, будет отображаться инверсным цветом (негативно).

**Сору.**

Как и **Cut**, команда **Сору** (копировать) помещает выделенный текст в буфер обмена, но, в отличие от **Cut**, **Copy** оставляет исходный текст на месте. Эта команда пригодится при копировании сложных участков текста программы или комментариев, которые требуются в нескольких местах программы.

**Paste.**

Команда **Paste** (вставить) используется, чтобы вставить содержимое буфера обмена туда, где в данный момент находится курсор. Вставлять из буфера обмена можно только то, что было ранее помещено туда командой **Cut** или **Сору**

**Delete.**

Команда **Delete** (удалить) уничтожает выделенный текст, не помещая его в буфер обмена. Для того чтобы выделить текст, поместите указатель мыши на первый из символов выделяемого текста и, удерживая левую кнопку нажатой, переместите мышь вправо/вниз до конца выделяемого участка. После этого текст, который вы выделили, будет отображаться инверсным цветом (негативно).

**Select All.**

Команда **Select All** (выделить все) используется, чтобы выделить все содержимое активного окна для последующей вырезки, копирования или удаления.

**16.Меню Edit. Find... Replace... Go To... InfoViewer Bookmarks... Bookmark. Breakpoints... Properties...**

Команды меню Edit (правка) позволяют быстро находить и исправлять текст в активном окне примерно так же, как это делается любым из распространенных текстовых процессоров.

**Find...**

Команда **Find...** (поиск) работает аналогично команде поиска в текстовом процессоре. Поскольку в языках С и C++ различаются прописные и строчные буквы, команду **Find...** можно настроить для поиска с учетом регистра, без учета регистра, а также поиска только полного слова. Команда **Find** также позволяет указать направление поиска (вперед или назад от текущей позиции курсора)

**Replace...**

Команда **Replace...** (заменить) вызывает окно диалога **Replace**, которое позволяет найти текст по образцу и заменить на заданный. Введите образец, который следует найти, затем укажите текст, на который его следует заменить, и наконец, выберите один из типов соответствия образцу. Можно задать соответствие с учетом регистра, без учета регистра, только полное слово или регулярное выражение.

**Go To...**

Поскольку приложение Windows обычно строится из десятков файлов, иногда не так уж просто найти место, где находится описание интересующего вас объекта. Это сделает для вас команда **Go To... (перейти к...).** Чтобы найти строку с описанием константы, переменной или функции, установите курсор на имя объекта (или непосредственно слева от него) и выполните эту команду. Интегрированная среда автоматически проследит, где находится описание данного имени.

**InfoViewer Bookmarks...**

На панели **InfoViewer** показано оглавление электронных книг. Вы можете вызвать любой раздел из этой иерархии.

**Bookmark.**

Команда **Bookmark** (закладка) устанавливает или удаляет закладку. Закладки используются для того, чтобы отмечать некоторые строки в тексте программы, к которым придется обратиться позже. Закладки можно установить как этой командой, так и при помощи параметра **Set Bookmark** (установить закладку) команды Find...

**Breakpoints...**

Команда **Breakpoints...** (точки останова) открывает окно диалога **Breakpoints**, позволяющее добавлять, удалять, выключать отдельные точки останова, а также удалить все точки сразу. Точки останова затем могут быть использованы командой **Go (запустить).**

**Properties...**

Команда **Properties...** (свойства) вызывает окно диалога, посредством которого задаются характеристики текущего окна. К примеру, файл в активном окне можно пометить "только для чтения" или указать, что он может содержать конструкции С, но не C++.

**17.Меню View. ClassWizard... Resource Symbols... Resource Includes... Full Screen. Toolbars... InfoViewer Query Results. InfoViewer History List. Project Workspace.**

Меню **View (просмотр)** обеспечивает доступ к командам, применяемым для отображения текущего проекта в разных видах. Сюда входят средства просмотра файлов справки, электронных книг, а также другие средства, облегчающие отладку приложений.

**ClassWizard...**

Команда **ClassWizard...** (мастер классов) вызывает средство высокоуровневого программирования, которое позволяет объявлять новые классы на основе компонент **Microsoft Foundation Class (MFC)**, либо добавлять новые методы обработки сообщений в существующий объект, построенный на основе MFC.

**Resource Symbols...**

Вы можете использовать команду **Resource Symbols...** (символы ресурсов) для отображения списка символов, используемых в данном файле ресурсов.

**Resource Includes...**

Обычно **Microsoft Developer Studio** (мастерская разработчика) хранит все ресурсы в файле с расширением RC, а определения символов — в файле RESOURCE.H. При помощи команды **Resource Includes...** (включаемые файлы ресурсов) вы можете изменить распределение ресурсов в файлах

**Full Screen.**

При помощи команды **Full Screen** (полный экран) вы можете развернуть текстовый редактор (или другие редакторы ресурсов) на полный экран.

**Toolbars...**

Команда **Toolbars...** (панели инструментов) вызывает появление окна диалога Toolbars, позволяющее выбрать панели инструментов, которые должны отображаться в среде Visual C++.

**InfoViewer Query Results.**

Команда **InfoViewer Query Results** (результаты запросов **InfoViewer**) показывает список предыдущих запросов на поиск информации в справочной системе.

**InfoViewer History List.**

Команда **InfoViewer History List** (хронологический список **InfoViewer**) показывает список разделов справки, к которым вы недавно обращались.

**Project Workspace.**

Команда **Project Workspace** (рабочее пространство проекта) выдает окно с закладками, соответствующими активным окнам текущего проекта

**18. Меню View. Info Viewer Topic. Output. Watch. Variables. Registers. Memory. Call Stack. Disassembly. Меню Insert.**

**Info Viewer Topic.**

Команда **InfoViewer Topic** (раздел **InfoViewer**) показывает содержание текущего раздела справочной системы.

**Output.**

Команда **Output (вывод)** активизирует окно **Output** для данного проекта.

**Watch.**

Команда **Watch (наблюдение)** отображает окно отладчика **Watch.** Это окно используется для наблюдения за значениями переменных по ходу выполнения программы.

**Variables.**

Команда **Variables (переменные)** открывает окно **Variables,** позволяющее увидеть активные в данный момент переменные, локальные переменные, или указатель **this**.

**Registers.**

Команда **Registers** вызывает окно регистров. Это окно позволяет считывать (но не модифицировать) состояние процессора, в том числе содержимое регистров и состояние флагов.

**Memory.**

Команда **Memory (память)** открывает окно, отображающее содержимое памяти с заданного адреса. Окно дает возможность просматривать состояние памяти в пределах адресного пространства программы.

**Call Stack.**

Команда **Call Stack (стек вызовов)** вызывает соответствующее окно диалога. В нем показана последовательность вложенных вызовов функций до уровня, на котором находится текущая строка текста.

**Disassembly.**

Команда **Disassembly (дизассемблер)** отображает окно с ассемблерным кодом, который был сгенерирован из ваших исходных текстов.

 Меню **Insert.**

Посредством меню **Insert (вставить)** вы сможете вставить файлы (Files...), ресурсы (Resources), копии ресурсов (Resource Copies), файлы в проект (Files into a Project), проекты в проект (Project) и компоненты рабочего пространства (Components...).

**19.Меню Build. Compile. Build. Rebuild All. Batch Build... Stop Build. Update All Dependencies.**

Меню **Build (сборка)** содержит много пунктов, необходимых для генерации исполняемого файла приложения (\*.ЕХЕ).

Команда **Compile (компиляция)** заставляет среду компилировать текст в текущем окне. Компиляция — важный этап разработки, поскольку именно здесь вы узнаете, содержит ли данный файл синтаксические ошибки. По этой причине существует возможность компилировать заголовочные файлы (\*.Н), несмотря на то, что они не могут исполняться. Если при компиляции обнаружатся ошибки (или предупреждения о возможных ошибках), сообщения о них будут помещены в окно Output.

**Build.**

Обычно программы на C/C++ состоят из множества файлов. Эти файлы могут поставляться с компилятором, с операционной системой, создаваться программистом или даже приобретаться у сторонних производителей.. В процессе сборки анализируются все файлы проекта и затем компилируются и компонуются лишь те из них, которые были изменены после создания исполняемого файла проекта.

Единственная разница между командами **Build** и **Rebuild All (полная сборка)** состоит в том, что **Rebuild All** не обращает внимания на даты изменения файлов и терпеливо компилирует и компонует все.

Команда **Batch Build... (пакетная сборка)** работает аналогично обычной сборке, но может создать в одном проекте сразу несколько целевых файлов.

Используйте команду **Stop Build (прервать сборку)**, если вы хотите остановить процесс сборки.

Команда **Update All Dependencies (обновить список зависимостей)** может быть использована, если вы хотите, чтобы система заново построила диски включаемых файлов для каждого из исходных файлов проекта.

**20.Меню Build. Debug. Execute. Settings... Configurations... Subprojects... Set Default Configuration...**

Меню **Build (сборка)** содержит много пунктов, необходимых для генерации исполняемого файла приложения (\*.ЕХЕ).

**Debug.**

Команда **Debug (отладка)** запускает интегрированный отладчик.

**Execute.**

Команда **Execute (выполнить)** используется для запуска вашей программы. В зависимости от типа целевого файла Visual C++ автоматически вызовет среду MS-DOS, Windows 95 или Windows NT для тестирования вашей программы.

**Settings...**

Команда **Settings... (параметры)** используется для выбора конфигурации проекта.

**Configurations...**

Команда **Configurations...(конфигурации)** служит для управления конфигурациями проекта.

**Subprojects...**

**Microsoft Developer Studio** позволяет вам включать в проект другие проекты. Доступ к ним осуществляется при помощи команды **Subprojects... (подпроекты).**

**Set Default Configuration...**

В большинстве случаев достаточно настроить конфигурацию для всего проекта в целом, но при необходимости вы можете задать для конкретных файлов проекта другие значения каких-либо параметров. Таким образом, конфигурация имеет иерархическую структуру: параметр, заданный на уровне проекта, относится ко всем файлам, кроме тех, для которых он переопределен на уровне файла. Например, если при помощи команды **Set Default Configuration (установить основную конфигурацию)** вы установили параметр **Default optimization (стандартная оптимизация),** то эта установка будет действовать на все файлы в данной конфигурации. Тем не менее, вы можете установить конкретные настройки оптимизации (или отключить ее вообще) для каких-либо файлов в отдельности. Значения параметров, заданные на уровне файла, будут использованы вместо тех, которые заданы на уровне проекта. Некоторые виды параметров (например, параметры компоновки) можно задавать только на уровне проекта.

**21.Меню Tools. Browse... Close Browse Info File. OLE Control Test Container. OLE Object View.**

**Меню Tools (инструменты)** обеспечивает доступ к множеству полезных вспомогательных средств интегрированной среды. Эти средства призваны облегчить, насколько это возможно, процесс разработки и модификации различных объектно-ориентированных приложений Windows.

**Browse...**

Команда **Browse... (найти)** выдаст хронологический список описаний объектов или ссылок на них, которые вы искали ранее. Список устроен наподобие стека, то есть при добавлении новый элемент становится первым в списке.

**Close Browse Info File**.

Команда **Close Browse Info File (закрыть файл информации браузера)** закрывает файл **Browse Info (информация браузера)**

**OLE Control Test Container**.

Эта команда вызывает утилиту **OLE Control Test Container (тестовый контейнер для управляющих элементов OLE)**. Это приложение, которое полностью поддерживает управляющие элементы OLE и позволяет встраивать их в свои окна и окна диалога.

**OLE Object View.**

Команда **OLE Object View (показать объект OLE)** вызывает **OLE Object Viewer (средство просмотра объектов OLE). OLE Object Viewer** — это вспомогательная программа для того, чтобы помочь разработчикам OLE-приложений лучше понять, что происходит в их системах.

**22.Меню Window. New Window. Split. Hide. Cascade.**

Меню **Window** позволяет управлять отображением различных окон, используемых в процессе разработки приложения. Это меню также позволяет устанавливать фокус ввода, указывая, какое окно сделать активным.

**New Window.**

Команда **New Window (новое окно)** дает еще одну возможность создать в текущем проекте новое окно

**Split**

Команда **Split (разделить)** делит текущее окно на две части.

**Hide.**

Команда **Hide (спрятать)** делает текущее окно скрытым.

**Cascade.**

Команда **Cascade (каскад)** располагает все открытые на экране окна одно за другим со сдвигом вправо и вниз, наподобие колоды карт.

**23.Меню Window. Tile Horizontally, Tile Vertically. Close All... Windows. Меню Help.**

Меню **Window** позволяет управлять отображением различных окон, используемых в процессе разработки приложения. Это меню также позволяет устанавливать фокус ввода, указывая, какое окно сделать активным.

**Tile Horizontally, Tile Vertically.**

Команды **Tile Horizontally (сверху вниз) и Tile Vertically (справа налево)** заставляют среду поровну разделить экран между всеми окнами. Преимущество такого расположения состоит в том, что вы можете одновременно видеть содержимое всех окон. Недостатки проявляются при большом количестве окон: тогда каждому из них достается место размером с почтовую марку.

**Close All...**

Команда **Close All... (закрыть все)** закрывает все открытые окна.

**Windows.**

Команда **Windows (окна)** открывает список всех имеющихся в среде окон, позволяя быстро найти нужное.

**Меню Help.**

Возможности такой разнообразной, насыщенной и интеллектуальной среды, как Visual C++, могут остаться невостребованными без последнего, но самого важного меню **Help (справка).** Пытаетесь ли вы разобраться в конструкции языка C/C++, узнать о средстве из среды Visual C++ или научиться использовать их совместно, вам помогут команды, доступные посредством меню **Help.**

**24.История языка С. Взаимоотношения с другими языками. Достоинства языка С. Малый размер. Набор команд языка. Быстродействие. Язык со слабой типизацией. Структурированный язык. Поддержка модульного программирования.**

Язык программирования Си был разработан и реализован в 1972 году сотрудником фирмы AT&T Bell Laboratories Денисом Ритчи. Прообразом языка Си для Д. Ритчи послужил язык Би, разработанный Кеном Томпсоном. Он является результатом эволюционного развития языков BCPL (Richards, M., "BCPL: A. Tool for Compiler Writing and System Programming", Proc. AFIPS SJCC, 34, 557-566, 1969) и Би (Johnson, S. C., and B. W. Kernighan, "The Programming Language B", Comp. Sci. Tech. Rep. No. 8, Bell Laboratories. 1973). Основным достоинством языка Си по сравнению с языками BCPL и Би является введение в него типов данных. Язык Си был разработан во время создания операционной системы UNIX (OC UNIX). Развитие языка Си продолжалось и после окончания его разработки и касалось, в частности, проверки типов данных и средств, облегчающих перенос программ в другую среду.

**Достоинства языка С**

**Оптимальный размер программы**

В основу С положено значительно меньше синтаксических правил, чем у других языков программирования. В результате для эффективной работы компилятора языка достаточно всего 256 Кб оперативной памяти. Действительно, список операторов и их комбинаций в языке С обширнее, чем список ключевых слов.

**Сокращенный набор ключевых слов**

Первоначально, в том виде, в каком его создал Деннис Ритчи, язык С содержал всего 27 ключевых слов. В ANSI С было добавлено несколько новых зарезервированных слов. В Microsoft С набор ключевых слов был еще доработан, и общее их число превысило 50. Многие функции, представленные в большинстве других языков программирования, не включены в язык С. Например, в С нет встроенных функций ввода/вывода, отсутствуют математические функции (за исключением базовых арифметических операций) и функции работы со строками. Но в то же время их легко можно переписать без ущерба для структуры программы — это безусловное преимущество С.

**Быстрое выполнение программ**

Программы, написанные на С, отличаются высокой эффективностью. Благодаря небольшому размеру исполняемых модулей, а также тому, что С является языком достаточно низкого уровня, скорость выполнения программ на языке С соизмерима со скоростью работы их ассемблерных аналогов.

**Упрощенный контроль за типами данных**

В отличие от языка Pascal, в котором ведется строгий контроль типов данных, в С понятие типа данных трактуется несколько шире. Это унаследовано от языка В, который так же свободно обращался с данными разных типов. Язык С позволяет в одном месте программы рассматривать переменную как символ, а в другом месте — как ASCII-код этого символа, от которого можно отнять 32, чтобы перевести символ в верхний регистр.

**Реализация принципа проектирования "сверху вниз"**

Язык С содержит все управляющие конструкции, характерные для современных языков программирования, в том числе инструкции for, if/else, switch/case, while и другие. На момент появления языка это было очень большим достижением. Язык С также позволяет создавать изолированные программные блоки, в пределах которых переменные имеют собственную область видимости. Разрешается создавать локальные переменные и передавать в подпрограммы значения параметров, а не сами параметры, чтобы защитить их от модификации.

**Модульная структура**

Язык С поддерживает модульное программирование, суть которого состоит в возможности раздельной компиляции и компоновки разных частей программы. Например, вы можете выполнить компиляцию только той части программы, которая была изменена в ходе последнего сеанса редактирования. Если бы язык С не поддерживал модульное программирование, то после внесения небольших изменений в программный код пришлось бы компилировать всю программу целиком, что могло бы занять слишком много времени.

**25.История языка С. Простой интерфейс с ассемблерными подпрограммами. Поразрядная обработка. Переменные-указатели. Гибкие структуры. Эффективность использования памяти. Переносимость. Библиотеки специальных функций.  
Простой интерфейс с ассемблерными подпрограммами.**В большинстве компиляторов С существует хорошо описанный метод вызова ассемблерных подпрограмм. С учетом раздельной компиляции и компоновки, эта возможность делает язык С очень сильным инструментом для приложений, требующих совмещения программ высокого уровня и ассемблерных программ. На многих системах процедуры, написанные на С, можно также интегрировать в ассемблерные программы.   
**Поразрядная обработка.**   
Зачастую при системном программировании необходимо манипулировать с объектами на уровне двоичных разрядов. Естественно, что язык С, связанный происхождением с операционной системой UNIX, располагает большим набором команд поразрядной обработки.   
**Переменные-указатели.**   
Одним из требований, предъявляемых к языку операционной системой, является возможность адресации определенной области памяти. Эта возможность также ускоряет выполнение программы. В языке С для этих целей служат указатели. Хотя указатели имеются и в других языках, С отмечен за возможность выполнять арифметические действия с указателями. Например, если переменная index указывает на первый элемент массива student\_records, значение index+1 будет адресом второго элемента массива student\_records.   
**Гибкие структуры.**   
Все массивы в С — одномерные. Многомерные структуры создаются на базе этих одномерных массивов. Массивы и структуры (записи) могут объединяться любым необходимым образом, образуя базы данных, ограничиваемые только возможностями программиста.   
**Эффективность использования памяти.**Во многом по тем же причинам, по которым программы С работают быстро, эти же программы очень эффективно используют память. Отсутствие утроенных функций позволяет отказаться в программах от тех функций, которые не требуются в данном приложении.   
**Переносимость.**   
Переносимость — это возможность переноса некоторой программы, работающей на одном компьютере или в одной операционной системе, на другой компьютер или в другую операционную систему. Программы, написанные на С, являются самыми мобильными в современном компьютерном мире. В особенности, это относится к мини- и микрокомпьютера.   
**Библиотеки специальных функций.**Для всех популярных компиляторов С имеется множество коммерчески доступных библиотек функций. Имеются библиотеки функций для графики, работы с файлами, поддержки баз данных, экранных окон, ввода данных, коммуникаций и общего назначения.Используя эти библиотеки, можно значительно сократить время проектирования.

**26.История языка С. Недостатки языка С. Слабая типизация. Отсутствие проверок на этапе исполнения. Использование языка Си. Будущее языка Си.**

Язык программирования Си был разработан и реализован в 1972 году сотрудником фирмы AT&T Bell Laboratories Денисом Ритчи. Прообразом языка Си для Д. Ритчи послужил язык Би, разработанный Кеном Томпсоном. Он является результатом эволюционного развития языков BCPL (Richards, M., "BCPL: A. Tool for Compiler Writing and System Programming", Proc. AFIPS SJCC, 34, 557-566, 1969) и Би (Johnson, S. C., and B. W. Kernighan, "The Programming Language B", Comp. Sci. Tech. Rep. No. 8, Bell Laboratories. 1973). Основным достоинством языка Си по сравнению с языками BCPL и Би является введение в него типов данных. Язык Си был разработан во время создания операционной системы UNIX (OC UNIX). Развитие языка Си продолжалось и после окончания его разработки и касалось, в частности, проверки типов данных и средств, облегчающих перенос программ в другую среду.

**Недостатки языка С**

**Упрощенный контроль за типами данных!**

То, что язык С не осуществляет строгого контроля за типами данных, можно считать как достоинством, так и недостатком. В некоторых языках программирования присвоение переменной одного типа значения другого типа воспринимается как ошибка, если при этом явно не указана функция преобразования. Благодаря этому исключается появление ошибок, связанных с неконтролируемым округлением значений. Как было сказано выше, язык С позволяет присваивать целое число символьной переменной и наоборотю. Побочными эффектами называются неконтролируемые изменения значений переменных. Поскольку С не осуществляет строгого контроля за типами данных, это дает большую гибкость при манипулировании переменными. Например, в одном выражении оператор присваивания (=) может использоваться несколько раз. Если бы С требовал однозначного определения типа данных, то это устранило бы появление побочных эффектов и неожиданных результатов, но в то же время сильно уменьшило бы мощь языка, сведя его к обычному языку высокого уровня.

**Ограниченные средства управления ходом выполнения программы**

Вследствие того, что выполнение программ на языке С слабо контролируется, многие их недостатки могут остаться незамеченными. Например, во Время выполнения программы не поступит никаких предупреждающих сообщений, если осуществлен выход за границы массива. Это та цена, которую пришлось заплатить за упрощение компилятора ради его быстроты и эффективности использования памяти.

**27.Исходные файлы и выполняемые файлы. Принципы программирования. Стандарт ANSI С. Эволюция языка C++ и объектно-ориентированное программирование. История C++. Использование объектов C++ для быстрого создания программы.**

[Принципы программирования](#_3.Основные_принципы_алгоритмизации)

**ANSI C** — стандарт [языка C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), опубликованный Американским национальным институтом стандартов ([ANSI](https://ru.wikipedia.org/wiki/%D0%90%D0%BC%D0%B5%D1%80%D0%B8%D0%BA%D0%B0%D0%BD%D1%81%D0%BA%D0%B8%D0%B9_%D0%BD%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%B8%D0%BD%D1%81%D1%82%D0%B8%D1%82%D1%83%D1%82_%D1%81%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BE%D0%B2)). Следование этому стандарту помогает создавать легко [портируемые](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) [программы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0).

Первый стандарт языка C был опубликован американским институтом ANSI. Через некоторое время он был принят международной организацией по стандартизации [ISO](https://ru.wikipedia.org/wiki/ISO), продолжившей выпускать следующие версии стандарта, которые стали приниматься как стандарт и институтом ANSI. Несмотря на это стандарт до сих пор чаще называют ANSI C, а не ISO C.

**Истоки C++**

История создания C++ начинается с языка С. И немудрено: C++ построен на фундаменте С. C++ и в самом деле представляет собой супермножество языка С. (Все компиляторы C++ можно использовать для компиляции С-программ.) C++ можно назвать расширенной и улучшенной версией языка С, в которой реализованы принципы объектно-ориентированного программирования

***Рождение C++***

Итак, C++ появился как ответ на необходимость преодолеть еще большую сложность программ. Он был создан Бьерном Страуструпом (Bjarne Stroustrup) в 1979 году в компании Bell Laboratories (г. Муррей-Хилл, шт. Нью-Джерси). Сначала новый язык получил имя "С с классами" (С with Classes), но в 1983 году он стал называться C++.

***Эволюция C++***

С момента изобретения C++ претерпел три крупных переработки, причем каждый раз язык как дополнялся новыми средствами, так и в чем-то изменялся. Первой ревизии он был подвергнут в 1985 году, а второй — в 1990. Третья ревизия имела место в процессе стандартизации, который активизировался в начале 1990-х. Специально для этого был сформирован объединенный ANSI/ISO-комитет, который 25 января 1994 года принял первый проект предложенного на рассмотрение стандарта

**28.Исходные файлы и выполняемые файлы. Некоторые усовершенствования по сравнению с языком С. Комментарии.**

**Имена перечисляемых типов. Имена структуры или класса. Блочные объявления. Операция уточнения области действия (scope). Описатель const. Анонимные объед**

**Комментарии**

Символ комментария //, действующий до конца строки, является теперь составной частью языка, а не только расширением Microsoft, как это было для языка С. Комментарии предназначены для упрощения чтения исходного кода программы путем добавления пояснений на литературном языке (Как правило, английском) в исходный код

//Это строка-комментарий, открывается //, закрывания не требует /\*

Это блок-комментарий, /\* - открывает комментарий, \*/ - закрывает \*/

**Имена перечислений, структур и объединений**

В C++ имя перечисления, структуры или объединения является именем типа. Это упрощает нотацию, поскольку не нужно использовать ключевые слова enum, struct и union перед именем соответствующего типа. Таким образом, в C++ выражения вида

struct Fruits{

// Компоненты структуры

};

Fruits stApple; // Определение переменной типа Fruits .

являются допустимыми.

**Блочные объявления**

В C++ допускаются объявления внутри блоков и после программных операторов, что позволяет определять (объявлять) объект в том месте программы, где он используется первый раз — во многих случаях это улучшает читабельность программы.

void myFunc()

{

int nFirstVar;

nFirstVar++;

// Другие операторы функции

...

float fSecondVar =7.0;

...

// Остальные операторы функции

}

Кроме того, можно определить и инициализировать переменную непосредственно внутри формального описания управляющей структуры:

for(int iCount = 0; iCount < MAX\_COUNT; iCount++) {...}

Оператор **scope (::)** уточняет область действия идентификатора переменной/функции/т.п

**например:**

namespace first

{ int x = 5; }

Int main () {

int y = first::x;//Правильно, программа видит переменную х, принадлежащую first

x= y-1;//Неправильно, программа не видит переменную х

}

Описатель **const:** Может быть добавлен при объявлении & инициализации переменной, значение которой программа не будет изменять. Например: const double pi = 3.141529; Альтернатива – директива препроцессора #define, будет выглядеть как #define pi 3.141529. #define быстрее, const надежнее .

**29.Исходные файлы и выполняемые файлы. Явное преобразование типов. Объявления функций.**

**Перегруженные функции. Значения параметров функций по умолчанию. Функции с неуказанным числом параметров. Ссылочные параметры функции. Операторы new и delete. Указатели voi**

**Явное преобразование типов**: Это преобразование, при котором мы явно выразили желание изменить тип данных. Как и любое преобразование, может привести к потере данных – в данном случае подразумевается, что это учтено разработчиком. Пример:  
double a = 3.5;  
int (a); //Производится явное преобразование типа переменной а

**Объявление функций:** В С++ в выражениях могут быть использованы только объявленные идентификаторы, поэтому любая функция должна быть объявлена до её вызова.  
**Перегрузка функций:** Несколько функций может иметь одинаковый идентификатор (имя), если параметры их вызова различаются в числе и/или типе; Программа при вызове функции под таким именем выберет ту из функций, которой соответствуют переданные аргументы.

**Значение параметров функций по умолчанию:** При описании параметров функции можно задать опциональный параметр, что позволяет, к примеру, при вызове функции с тремя параметрами задать только два аргумента. Для этого необходимо для последнего параметра задать значение по умолчанию. Например:

int divide (int a, int b=2){return a/b;}

**Функции с неуказанным числом параметров:** Если в функцию может быть передано неопределенное число параметров одного **типа int average (int n, ..)** {~~~~}   
**Ссылочные параметры функции**: Параметр, передаваемый в функцию, может быть передан двумя путями – можетббыть передано значение - тогда функция работает с локальной копией параметра, или может быть передана ссылка на значение (= Адрес этого значения), и тогда функция работает непосредственно с оригинальным параметром.

**Операторы New & Delete** служат для динамического выделения памяти в ходе выполнения программы в С++; При помощи оператора **New** программа выделяет память и возвращает адрес выделенной памяти, при помощи оператора **Delete** программа освобождает выделенную память.

В C++ существует специальный тип указателя, который называется указателем на неопределённый тип.

Для определения такого указателя вместо имени типа используется ключевое слово **void** в сочетании с описателем, перед которым располагается символ ptrОперации \*.

**30.Исходные файлы и выполняемые файлы. Основные усовершенствования по сравнению с языком С.(часто повторяется). Конструкторы классов и инкапсуляция данных. Класс struct. Конструкторы и деструкторы. Сообщения. "Дружественные" классы**.

**Конструкторы и деструкторы:**

Это функции класса. Конструктор при вызове создает и инициализирует объект класса (Что позволяет избежать обращения к объекту до его инициализации), деструктор при вызове очищает память, выделенную под объект (Сам объект удаляется в последнюю очередь).

**Инкапсуляция:**

Для любого поля класса (Переменная, функция, т.д.) можно задать параметры доступа к нему. Возможны 2 варианта – Public (Общедоступен) и Private (Доступен только для объектов этого класса). Инкапсуляция поля есть запрещение доступа к нему извне.

**Дружественные классы и функции:**

Иногда нам надо дать доступ к инкапсулированным данным какой-то функции или классу извне (Например, чтобы не описывать эту же функцию еще раз.) В таком случае в описании класса мы записываем friend class имя\_класса; или friend прототип\_функции;, и тогда этот класс или функция получают доступ к инкапсулированным данным.

**Класс struct:**

Класс, созданный с помощью **struct,** практически не отличается от обычного класса (Объявленного через class). Единственное функциональное отличие – в объекте вида class по умолчанию любой член может быть вызван только другим членом класса (private), а в объекте вида struct любой член по умолчанию доступен кому угодно (public).

**31. Исходные файлы и выполняемые файлы. Перегрузка операций. Производные классы. Полиморфизм при использовании виртуальных функций. Библиотеки потоков. Базовые элементы программы на С. Пять основных компонентов программы.**

**Перегрузка операций**

Так же, как и для функций, язык C++ позволяет переопределить действие большинства операций так, чтобы при использовании с объектами конкретного класса они выполняли заданные действия. Такая перегрузка желательна с точки зрения непротиворечивости использования. В C++ механизм перегрузки операций реализован путем определения функции с ключевым словом **operator**, стоящим перед перегружаемой операцией.

В случае переопределения операции с помощью метода класса, последний в качестве неявного параметра принимает ключевое слово **this**, являющееся указателем на данный объект класса. Поэтому если переопределяется бинарная операция, то переопределяющий ее метод класса должен принимать только один параметр, а если унарная, то метод класса вообще не имеет параметров.

Если операция переопределяется при помощи дружественной функции, то она должна принимать соответственно два и один параметр.

При переопределении операций необходимо учитывать некоторые имеющиеся в языке ограничения:

* нельзя изменять количество параметров операции;
* нельзя изменить приоритет операций;
* нельзя определить новые операции;
* нельзя использовать параметры по умолчанию;

**Полиморфизм**

Прямым результатом **наследования** явилось включение в **язык виртуальных функций**. **Наследование** — некоторый класс может передавать свои компоненты другому классу. При этом первый класс называется базовым, а второй — производным классом (или подклассом)..

Что же делать, чтобы иметь возможность вызывать оба метода? Ответ на этот вопрос дает **полиморфизм** — использование одного и того же вызова для ссылки на разные методы в зависимости от типа объекта.

**Библиотека потоков**

Библиотека потоков данных TPL модель потоков данных поддерживающая программирование на основе субъектов путем обеспечения в процессе передачи сообщений для не детализированного потока данных и задач по конвейеризации.

**Базовые элементы программы на С**

* Main Menu ( Главное меню)
* ToolBars (Панель инструментов)
* Symbol Window ( Окно символов)
* Symbol Panel (Панель символов)
* File Tab Bar (Панель вкладок файлов)
* Code Editor (Редактор кода)
* File Tree Window ()
* Messange Window (Окно сообщений)

**32.Написание и компиляция простых программ на C/C++. Написание вашей первой программы. Пример простой программы на С. Структура простой программы. Как сделать программу читаемой. Подготовка и компиляция простых программ на С/С++.**

**Компиляция** — 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), выполняемая компилятором

**Структура программы на языке C** Программа может состоять из одной или нескольких, связанных между собой, функций, главная из которых называется **main**– именно с нее начинается выполнение программы. Поэтому, наличие функции с таким именем в любой программе. Описание функции состоит из заголовка и тела. Заголовок в свою очередь состоит из директив препроцессора типа **#include** и т. д. и имени функции.

**НЕСКОЛЬКО СОВЕТОВ, КАК СДЕЛАТЬ ПРОГРАММУ ЧИТАЕМОЙ**

* выбор осмысленных обозначений для переменных
* использование комментариев
* помещать каждый оператор на отдельной строке
* использовании пустых строк ( для того, чтобы отделить одну часть функции, соответствующую некоторому семантическому понятию, от другой
* **Компиляция простых программ на С/С++**

Исполнить исходные файлы нельзя, их необходимо ***скомпилировать***, т.е. создать исполняемый файл, содержащий в себе инструкции процессора и пригодный для запуска на компьютере.

Процесс преобразования исходных файлов в исполняемый называется *компиляцией*. Если ваша программа состоит из одного исходного файла hello.c, то для его компиляции  достаточно выполнить команду:

bash$ gcc hello.c -o hello

В результате получится файл hello, имя которого мы указали в опции -o. Этот файл является исполняемым и его можно запускать (**exe**cute) при помощи команды:

bash$ ./hello

Пара символов ./ перед hello означает "искать исполняемый файл hello в текущей директории".

Строчка

bash$ gcc xxx.c yyy.c -o zzz -I./common -I.. -lm

**33.Написание и компиляция простых программ на C/C++. Редактирование текста программы. Сохранение программ. Построение программы.**

**Компиляция** — 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), выполняемая компилятором

**Сохранение файла**

Желательно сохранить файл до того, как вы приступите к его компиляции и компоновке, а тем более до того, как попытаетесь запустить программу на выполнение. Чтобы сохранить введенный только что код, вы можете либо щелкнуть на третьей кнопке слева на стандартной панели инструментов, либо выбрать в меню File команду Save, либо нажать [Ctrl+S]. Когда вы в первый раз выбираете команду Save, открывается диалоговое окно Save.

**Построить проекты Visual C++ можно двумя способами:**

* с помощью Visual Studio;
* с помощью командной строки.

При построении приложения Visual C++ в Visual Studio в диалоговом окне "Окна свойств" проекта можно изменить множество параметров построения.

Для программистов, которые предпочитают выполнять построение приложений из командной строки, в Visual C++ представлены средства командной строки. Для построения проекта Visual C++ можно использовать следующие средства командной строки:

* DEVENV.EXE (Команда Devenv предоставляет возможность установки из командной строки различных параметров для интегрированной среды разработки (IDE), а также для компиляции, построения и отладки проектов. Используйте эти переключатели для запуска IDE из файла сценария или из BAT-файла, например сценария построения программы в ночное время, либо для запуска IDE в особой конфигурации.)
* NMAKE.EXE (Утилита построения программ (Майкрософт) NMAKE.EXE — это средство, предназначенное для построения проектов на основании команд, содержащихся в файле описания.)
* Справочник по программе VCBUILD (Программа VCBUILD.exe может использоваться для построения проектов Visual C++ и решений Visual Studio из командной строки. Использование этого средства аналогично выполнению команд Построить проект или Построить решение в интерфейсе интегрированной среды разработки Visual Studio.)

**34.Написание и компиляция простых программ на C/C++. Использование утилиты Project Workspace. Создание нового проекта. Добавление файлов к проекту. Запуск команд Build или Rebuild All.**

**Компиляция** — 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), выполняемая компилятором

**Workspace**

Каталог Workspace содержит такие файлы проекта, как источники данных, страницы, утилиты и загрузки.

**Обзор**

Каталог Workspace содержит все файлы проекта Symphony. По умолчанию там сохраняются источники данных, события, страницы, утилиты и файлы загружаемые пользователями. Часто разработкичи используют данный каталог для хранения различных вспомогательных файлов, таких как CSS и JavaScript или картинок шаблонов.

**Использование**

Пользователи могут создавать любую структуру подкаталогов в каталоге workspace.

URL адрес каталога workspace включён в системные параметры Symphony.

**Детали**

По умолчанию система создаёт и использует четыре подкаталога в каталоге **workspace**:

/data-sources  
/events  
/pages  
/utilities

Файлы физически расположенные в данных подкаталогах могут быть отредактированны непосредственно в панели управления администратора.

При необходимости могут быть созданы дополнительные подкаталоги. На практике довольно часто создаются вспомогательные каталоги (такие, как /styles, /images и /scripts) и /uploadsдля хранения загружаемого пользовательского контента.

**Создание нового проекта**

При программировании на С++ существует понятие **проекта**. Проект может объединять в себя несколько файлов с текстом программы, но после компиляции проекта получается один exe-модуль. В консольном приложении работа проекта начинается с вызова функции main, а уже из нее идет вызов всех остальных функций, которые могут находиться в разных файлах.

Для создания нового проекта нужно выбрать пункт меню File->New, затем на закладке Projects выбрать тип проекта - Win32 Console Application, в поле Location написать путь к проекту (на своем диске!), в поле Project Name - имя проекта (например, test).

**Добавление файлов в проект**

После того как проект создан, в него можно сразу же добавлять новые файлы. Перейдите на вкладку FileView и щелкните правой кнопкой мыши на элементе ERRORfiles. Контекстное меню с выделенной командой Add Files to Project.... При выборе данной команды появляется окно InsertFilesintoProject, где вы можете отметить файлы, подлежащие включению в проект. Одно замечание по поводу типа файлов: файлы заголовков (с расширением Н) не включаются в список файлов проекта, а добавляются в проект непосредственно во время построения программы с помощью директив препроцессора #include. В нашем случае нужно найти созданный ранее файл ERROR.C и выполнить на нем двойной щелчок, в результате чего файл автоматически будет добавлен в проект.

**Запуск команд Build или Rebuild All.**

Команда Build (ей соответствует комбинация клавиш <Alt>+<F9>) строит проект, проверяя метки времени всех исходных файлов в проекте, поэтому если исходные файлы (или файлы, которые в них включены) являются более новыми, чем зависимые OBJ-файлы, то соответствующие модули проекта будут перекомпилированы.

Команда Rebuild All (ей соответствует комбинация клавиш <Ctrl>+<Alt>+<F9>) выполняет то же действие, что и  Build, причем все файлы будут повторно сгенерированы или откомпилированы и скомпонованы независимо от их меток времени.

**35.Написание и компиляция простых программ на C/C++. Отладка программы. Понимание сообщений об ошибках и предупреждений. Распространенная ошибка при использовании нового языка. Переключение между окном вывода сообщений и окном редактирования. Использование функций замены или быстрого поиска. Выбор опций замены**

**Компиляция** — 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), выполняемая компилятором

**Отладка** — этап разработки компьютерной программы, на котором обнаруживают, локализуют и устраняют ошибки. Чтобы понять, где возникла ошибка, приходится:

-узнавать текущие значения переменных;

-выяснять, по какому пути выполнялась программа.

**Типы ошибок при отладке:**  
1)предупреждения компилятора  
2)ошибки компилятора  
3)ошибки компоновщика

Предупреждения компилятора – несерьёзные ошибки, которые не препятствуют компиляции программы.

Ошибки — это условия, которые препятствуют завершению компиляции ваших файлов. Ошибки компилятора ограничены отдельными файлами исходного кода и являются результатом “синтаксических ошибок”. Например, выражение for(;).Ошибки компилятора всегда будут включать номер строки, в которой была обнаружена ошибка.

Ошибки компоновщика — это, проблемы с поиском определения функций, структур, классов или глобальных переменных, которые были объявлены, но не определены, в файле исходного кода.

**Существуют две взаимодополняющие технологии отладки:**

-**Использование отладчиков** — программ, которые включают в себя пользовательский интерфейс для пошагового выполнения программы: оператор за оператором, функция за функцией, с остановками на некоторых строках исходного кода или при достижении определённого условия.

-**Вывод текущего состояния программы** с помощью расположенных в критических точках программы операторов вывода — на экран, принтер, громкоговоритель или в файл. Вывод отладочных сведений в файл называется журналированием.

Переключение между окном вывода и окном редактирования происходит либо мышкой, либо с помощью комбинации клавиш Alt + Tab.

**Использование команд Find и Replace**

Довольно часто в процессе программирования возникают ситуации, когда вам нужно найти и заменить какое-то слово в тексте программы. Вы, конечно же, можете сделать это с помощью диалогового окна, открываемого командой Replace... из меню Edit, но имеется и более быстрый способ. Рассмотрите внимательно панель инструментов, , и найдите в поле списка Find слово continue. Чтобы воспользоваться этим средством поиска, щелкните мышью в поле и введите слово, которое хотите найти, после чего нажмите [Enter]. На рис. показан результат такого поиска. В тексте программы выделено слово continue, обнаруженное первым.

**36.Написание и компиляция простых программ на C/C++. Переключение между окном вывода сообщений и окном редактирования. Быстрый способ. Значение сообщений об ошибках и предупреждений. Повторная сборка программы ERROR.С. Запуск программы.**

**Компиляция** — 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), выполняемая компилятором

Переключение между окном вывода и окном редактирования происходит либо мышкой, либо с помощью комбинации клавиш Alt + Tab.

**Типы ошибок при отладке:**  
1)предупреждения компилятора  
2)ошибки компилятора  
3)ошибки компоновщика

Предупреждения компилятора – несерьёзные ошибки, которые не препятствуют компиляции программы. Предупреждения компилятора — это признак того, что что-то может пойти не так во время выполнения. Например, в условии =, а не == .  
Ошибки — это условия, которые препятствуют завершению компиляции ваших файлов. Ошибки компилятора ограничены отдельными файлами исходного кода и являются результатом “синтаксических ошибок”. Например, выражение for(;).Ошибки компилятора всегда будут включать номер строки, в которой была обнаружена ошибка.

Ошибки компоновщика — это, проблемы с поиском определения функций, структур, классов или глобальных переменных, которые были объявлены, но не определены, в файле исходного кода.

**Запуск программы**

Чтобы запустить программу, просто выберите в меню Project команду Execute.

Для Visual Studio: запуск программы с отладкой – F5, без отладки – Ctrl + F5.

**37.Написание и компиляция простых программ на C/C++. Использование встроенного отладчика. Использование команд пошагового выполнения (Step Into и Step Over). Определение точек останова (breakpoints). Запуск программы с точками останова. Использование быстрого просмотра**

**Компиляция** — 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), выполняемая компилятором.

**Использование встроенного отладчика**

К счастью, встроенный в VisualC++ отладчик содержит ряд средств, которые послужат для вас спасательным кругом в подобной ситуации. Во-первых, вы можете выполнять программу пошагово, строка за строкой. Во- вторых, вам предоставляется возможность анализировать значения переменных в любой момент выполнения программы

**Разница между командами Step Into и Step Over**

Когда вы начинаете процесс отладки, появляется панель инструментов Debug. Из множества представленных на ней кнопок наиболее часто задействуются Step Into (четвертая справа в верхнем ряду) и Step Over(третья справа). В обоих случаях программа будет запущена на выполнение в пошаговом режиме, а в тексте программы выделяется та строка, которая сейчас будет выполнена. Различия между командами Ste pInto и Step Over проявляются только тогда, когда в программе встречается вызов функции. Если выбрать команду Step Into, то отладчик войдет в функцию и начнет выполнять шаг за шагом все ее операторы. При выборе команды Step Over отладчик выполнит функцию как единое целое и перейдет к строке, следующей за вызовом функции. Эту команду удобно применять в тех случаях, когда в программе делается обращение к стандартной функции или созданной вами подпрограмме, которая уже была протестирована. Давайте выполним пошаговую отладку нашей программы.

**Точка останова** ­– это преднамеренное прерывание выполнения программы, при котором выполняется вызов отладчика

**Работа с точками останова**

Предположим, что вы поставили точку останова в строке программы, содержащей вызов функции scanf{). Теперь выберите команду Go— либо из меню, либо нажав клавишу [F5]. Обратите внимание, что выполнение программы прерывается не на первой строке программы, а на строке, содержащей точку останова. **Окно QuickWatch,**

Команда **QuickWatch...** открывает диалоговое окно **QuickWatch** , которое позволяет по ходу выполнения программы анализировать значения переменных. Простейший способ определить значение переменной с помощью данного окна состоит в том, что курсор помещается на имени переменной в окне редактирования, а затем нажимается комбинация клавиш [Shift+F9].

**38.Данные. Идентификаторы. Ключевые слова. Символы. Данные: переменные и константы. Данные: типы данных.**

**Данные** — это любая информация, представленная в формализованном виде и пригодная для обработки алгоритмом.

**Данные** делятся на **переменные** и **константы**.

**Переменные** — это такие данные, значения которых могут изменяться в процессе выполнения алгоритма.

**Константы** — это данные, значения которых не меняются в процессе выполнения алгоритма.

Типы данных принято делить на **простые (базовые)** и **структурированные**.

К основным базовым типам относятся:

· **целый (INTEGER)** — определяет подмножество допустимых значений из множества целых чисел;

· **вещественный (REAL, FLOAT)** — определяет подмножество допустимых значений из множества вещественных чисел;

· **логический (BOOLEAN)** — множество допустимых значений — истина и ложь;

· **символьный (CHAR)** — цифры, буквы, знаки препинания и пр.

**Структурированные** типы описывают наборы однотипных или разнотипных данных, с которыми алгоритм должен работать как с одной именованной переменной.

Наиболее широко известная структура данных — **Массив**. **Массив -**представляет собой упорядоченную структуру однотипных данных, которые называются элементами массива.

**Идентификаторы**

**Идентификаторами** называются имена, присваиваемые : переменным, константам, типам данных и функциям, используемым в программах. Компилятор распознает только первые 31 символ.

**Ключевые слова**

**Ключевые слова** являются встроенными идентификаторами, каждому из которых соответствует определенное действие. Изменить назначение ключевого слова **нельзя.** (С помощью директивы препроцессора **#define** можно создать "псевдоним" ключевого слова, который будет дублировать его действия, возможно, с некоторыми изменениями.) Помните, что имена идентификаторов, создаваемых в программе, не могут совпадать с ключевыми словами языков C/C++

**39.Три целочисленных типа. Описание данных целого типа. Целые константы. Инициализация переменных целого типа. Модификатор unsigned.**

**Три типа целых чисел В C/C++** поддерживаются три типа целых чисел. Наравне со стандартным типом int существуют типы shortint (короткое целое) и longint (длинное целое). Допускается сокращенная запись short и long. Хотя синтаксис самого языка не зависит от используемой платформы, размерность типов данных short, intи long может варьироваться. Гарантируется лишь, что соотношение размерностей таково: short <= int <= long. В Microsoft Visual C/C++ для переменных типа short резервируется 2 байта, для типов int и long — 4 байта

**Описание данных целого типа.**

При описании данных необходимо ввести только тип, за которым должен следовать список имен переменных. Ниже приведены некоторые возможные примеры описаний:

int erns;

short stops;

long Johns;

int hogs, cows, goats;

В качестве разделителя между именами переменных необходимо использовать запятую; весь список должен оканчиваться символом «точка с запятой». Вы можете собрать в один оператор описания переменных с одним и тем же типом или, наоборот, разбить одно описание на несколько операторов. Например, описание

int erns, hogs, cows, goats;

Иногда вам могут встретиться сочетания ключевых слов, как, например, long int или short int. Эти комбинации являются просто более длинной записью ключевых слов long и short.

**Целая константа**

Может быть записана в десятичной,восмеричной или шестнадцатиричной системе счисления.

В десятичной записывается как обычное десятичное число,при условии,что первая цифра не является нулём.

В восмеричной целая константа записывается восмеричными цифрами и должна начинаться с нуля.

В шестнадцатиричной целая константа записывается шестнадцатиричными цифрами и должна начинаться с символов 0x или 0X.При этом для обозначения цифр от 10 до 15 иcпользуются буквы A,B,C,D,F

**Модификаторы unsigned**

Компиляторы языков C/C++ позволяют при описании переменных некоторых типов указывать модификатор unsigned. В настоящее время он применяется с четырьмя типами данных: char, short, int и long. Наличие данного модификатора указывает на то, что значение переменной должно интерпретироваться как беззнаковое число, т.е. самый старший бит является битом данных, а не битом знака.