«Основы программирования»

Слайд 2

Тема 1. Базовые средства языка C++

Представляем Вам электронный учебник по курсу «Основы программирования». Курс разработан для проекта Росдистант.

В рамках этого курса мы будем изучать программирование на алгоритмическом языке C++ [си плюс плюс].

«Язык программирования C [си] был разработан в 1972 году Де́ннисом Ри́тчи как системный язык программирования, то есть язык для написания операционных систем. Основной задачей Ритчи было создание легко компилируемого минималистического языка. Этот язык предоставлял бы эффективный доступ к памяти, относительно быстро выполнялся, и на котором можно было бы писать эффективный код. Таким образом, при разработке высокоуровневого языка, был создан язык C [си], который во многом относился к языкам низкого уровня, оставаясь при этом независимым от платформ, для которых мог быть написан код.

Язык C [си] имеет ряд существенных особенностей, которые выделяют его среди других языков программирования. В значительной степени на формирование идеологии языка повлияла цель, которую ставили перед собой его создатели – обеспечение системного программиста удобным инструментальным языком, который мог бы заменить Ассемблер.

C [си] в конечном итоге стал настолько эффективным и гибким, что в 1973 году Деннис Ритчи и Кен То́мпсон переписали больше половины операционной системы UNIX [юникс], используя этот язык. Многие предыдущие операционные системы были написаны на языке ассемблера. Язык C [си] и операционная система UNIX [юникс] тесно связаны между собой, и популярность первого отчасти связана с успехом второго.

В 1978 году Бра́йан Ке́рниган и Де́ннис Ритчи опубликовали книгу – «Язык программирования С [си]». Эта книга стала стандартом и своеобразной инструкцией к С [си]. Когда требовалась максимальная портативность, то программисты придерживались рекомендаций из этой книги, поскольку большинство компиляторов в то время были реализованы в соответствии со стандартами, присутствующими в этой книге» [1].

Слайд 3

Язык программирования С ++ [си плюс плюс] был разработан Бьёрном Стра́уструпом в качестве дополнения к языку С [си] в 1979 году. Он добавил множество новых особенностей в язык С [си], который был одобрен международной организацией по стандартизации в 1998 году и повторно в 2003 году, под названием С ++03 [си плюс плюс ноль три]. Потом были еще три обновления, одобренные в 2011, 2014 и 2017 годах, соответственно, которые добавили больше функциональных возможностей.

Язык программирования С ++ [си плюс плюс] был разработан на основе языка С [си], и является расширением этого языка, поэтому программы написанные на языке С [си], могут обрабатываться компилятором С ++ [си плюс плюс].

«Смысл философии языков С [си], и С ++ [си плюс плюс] можно определить выражением – доверять программисту. Например, компилятор не будет вам мешать сделать что-то новое, что имеет смысл, но также не будет мешать вам сделать что-то такое, что может привести к сбою. Это одна из главных причин, почему так важно знать то, что вы не должны делать, как и то, что вы должны делать, создавая программы на языках С [си], и С ++ [си плюс плюс].

Недавно был собран комитет по стандартам языка, чтобы завершить работу над последней версией языкового стандарта, С ++ 20 [си плюс плюс

двадцать] и определить новую функциональность, которая появится в следующем крупном релизе.

После С ++ 17 [си плюс плюс семнадцать] – это будет шестая редакция стандарта. Этот язык прошёл долгий путь от своего бытия над множеством С [си]. Программирование на С ++ [си плюс плюс] включает в себя множество стилей: от простейшего С [си] с классами, до написания кода, который выглядит как произведение искусства» [1].

Слайд 4

Перед тем как приступить к изучению программирования на языке С++ [си плюс плюс], рассмотрим технологии программирования и методы программирования. На слайде приведено определение технологии программирования.

В технологии должны быть определены: последовательность выполнения операций, условия, при которых выполняется каждая операция, описание самих операций – исходные данные, нормативные документы, в том числе стандарты, критерии и методы оценки, результаты и др.

В зависимости от метода, программирование подразделяют:

– на процедурное – метод, в соответствии с которым программы пишутся как перечни последовательно выполняемых команд. При этом используются процедурно-ориентированные языки программирования, например, PL [пи эль], Паскаль, С [си];
– структурное, модульное – метод написания программ небольшими независимыми структурированными частями, модулями, каждый из которых связан с какой-либо функцией. Основу структурного метода программирования составляет декомпозиция создаваемой системы на отдельные функции и задачи, которые, в свою очередь, разбиваться на более мелкие. Процесс декомпозиции продолжается вплоть до

определения конкретных процедур. Результирующая программа организуется в виде совокупности взаимосвязанных по определенным правилам модулей. Это упрощает разработку сложных программных продуктов и их тестирование. Языки для модульного программирования – TurboPascal [Турбо Паскаль], C ++ [си плюс плюс], Ада;

– декларативное – метод, предназначенный для решения задач искусственного интеллекта. В указанном контексте программа описывает логическую структуру решения задачи, указывая преимущественно, что нужно сделать, не вдаваясь в детали. Используются языки программирования типа Пролог.

Слайд 5

«Объектно-ориентированное программирование представляет собой стратегию проектирования программных средств из объектов. Объект – это нечто, способное пребывать в различных состояниях и имеющее множество операций. Состояние определяется как набор атрибутов объекта. Операции, связанные с объектом, предоставляют сервисы другим объектам для выполнения определенных вычислений. Объекты объединяются в классы, каждый из которых имеет описания всех атрибутов и операций. Объектно-ориентированное программирование – это метод, основанный на использовании концепции объекта, абстрагирующего конкретные его реализации в предметной области. При этом данные тесно связываются с выполняемыми над объектами процедурами.

Функциональное программирование – это метод, основанный на разбиении алгоритма решения задачи на отдельные функциональные модули, а также описании их связей и характера взаимодействия.

Эвристическое программирование – это метод, основанный на моделировании мыслительной деятельности человека. Используется для

решения задач, не имеющих строго формализованного алгоритма или связанных с неполнотой исходных данных.

Компонентное программирование – это метод создания программного обеспечения путем сбора объектов-компонентов в библиотеки или исполняемые файлы. Объекты рассматриваются на логическом уровне проектирования программной системы, а компоненты – как непосредственная физическая реализация объектов. Один компонент может быть реализацией нескольких объектов или даже некоторой части объектной системы, полученной на уровне проектирования. Компоненты в отличие от объектов могут изменяться и пополняться новыми функциями и интерфейсами. Они конструируются в виде некоторой программной абстракции, состоящей из трех частей: информационной, внешней и внутренней» [1].

Слайд 6

Решение задачи на компьютере проходит по этапам, указанным на слайде.

«Любая задача начинается со словесного описания, которое называется условием задачи, а в программировании – спецификацией задачи. Далее условие следует формализовать, то есть так записать спецификацию, чтобы можно было ее решить на компьютере. Формализацией условия начинается этап, который называется математической формулировкой задачи. Выводятся формулы и выбираются методы решения задачи, то есть строится математическая модель задачи. На этом этапе вырабатывается точная формулировка цели задачи, определяется роль компьютера в ее реализации и целесообразность применения компьютера вообще. От корректности постановки задачи пользователем в значительной мере зависят все последующие этапы и успех решения задачи в целом.

Далее в виде алгоритма строится последовательность стандартных действий, выполнение которых даст искомый результат. На выбранном языке программирования проектируется и пишется программа для компьютера. Затем выполняется ее тестирование и отладка на компьютере. Выполняются необходимые расчеты. На слайде приведены этапы решения задач на компьютере.

Осуществляется формализация описания задачи, то есть с помощью математических, статистических и других методов соотношения между величинами выражаются с помощью математических, логических формул. Представление объекта исследования, задачи в виде совокупности математических отношений называется его математической моделью.

Приступая к построению модели, обычно вначале анализируют имеющийся опыт решения сходных задач, выясняют, какие методы формализации больше всего подходят для решения поставленной задачи. Метод решения поставленной задачи будет вытекать из принятой математической модели. Если задача может быть решена различными методами, выбирается тот, который наилучшим образом удовлетворяет ее основным требованиям» [2].

Слайд 7

«Компьютеры понимают только очень ограниченный набор инструкций, и чтобы заставить их что-то делать, нужно четко сформулировать задание, используя эти же инструкции. Программа – это набор инструкций, которые указывают компьютеру, что ему нужно делать. Физическая часть компьютера, которая выполняет эти инструкции, называется «железом» или аппаратной частью.

Процессор компьютера не способен понимать напрямую языки программирования. Очень ограниченный набор инструкций, которые

изначально понимает процессор, называется машинным кодом, или машинным языком.

Для решения проблем читабельности кода и чрезмерной сложности были разработаны высокоуровневые языки программирования. Они позволяют писать и выполнять программы, не переживая о совместимости кода с разными архитектурами процессоров» [3].

Программы, написанные на языках высокого уровня, также должны быть переведены в машинный код перед выполнением.

Процесс перевода программного кода с алгоритмического языка на язык машинных команд называется трансляция.

Существует два способа трансляции: компиляция и интерпретация.

Термины «компилятор» и «интерпретатор» описывают способ, с помощью которого выполняется программа. Компилятор – это программа, которая читает код и создаёт автономную, способную работать независимо от другого аппаратного или программного обеспечения, исполняемую программу. Такую программу процессор понимает напрямую. При запуске программы весь код компилируется целиком, а затем создаётся исполняемый файл, и уже при повторном запуске программы компиляция не выполняется.

Процесс компиляции показан на слайде.

Слайд 8

«Ли́нкинг – это процесс связывания всех объектных файлов, генерируемых компилятором в единую исполняемую программу, которую вы затем сможете запустить, выполнить. Это делается с помощью программы, которая называется ли́нкер, компоновщик.

Кроме объектных файлов, линкер также подключает файлы из Стандартной библиотеки C ++ [си плюс плюс] или любой другой библиотеки, которую вы используете, например, из библиотеки графики или

звука. Сам по себе язык С ++ [си плюс плюс] довольно маленький и простой. Тем не менее, к нему подключается большая библиотека дополнительных функций, которые могут использовать ваши программы, и эти функции находятся в Стандартной библиотеке С ++ [си плюс плюс].

После того, как компоновщик закончит ли́нкинг всех объектных файлов при условии, что не будет ошибок, вы получите исполняемый файл» [4].

Интерпретатор – это программа, которая напрямую выполняет код без его предыдущей компиляции в исполняемый файл.

Интерпретаторы более гибкие, но менее эффективные, так как процесс интерпретации выполняется повторно при каждом запуске программы. Интерпретаторы бывают очень полезны, когда надо быстро проверить какую-то идею, для отладки или во время обучения новому языку.

Процесс интерпретации показан на слайде. Интерпретатор построчно читает исходный код программы и выполняет инструкции, содержащиеся в текущей строке, потом переходит к следующей строке. Когда используется интерпретатор, он должен присутствовать все время для выполнения программы.

Любой язык программирования может быть компилируемым или интерпретируемым. Однако такие языки, как С [си], С++ [си плюс плюс] и Pascal [паскаль], – компилируются, в то время как скриптовые языки, такие как Perl [перл] и Java Script [джава скрипт], – интерпретируются.

Некоторые языки программирования, например, Java [джава], могут как компилироваться, так и интерпретироваться.

Слайд 9

Обычный разговорный язык состоит из четырех основных элементов: символов, слов, словосочетаний и предложений. Алгоритмический язык

содержит подобные элементы, только слова называют элементарными конструкциями, словосочетания – выражениями, предложения – операторами.

Входом компилятора служит программа на языке программирования. С точки зрения компилятора, это просто последовательность символов. Задача первой фазы компиляции, лексического анализатора, заключается в разборе входной цепочки и выделении некоторых более крупных единиц, лексем. Примерами лексем являются основные ключевые слова, идентификаторы, константные значения и так далее. На этапе лексического анализа обычно также выполняются такие действия, как удаление комментариев.

Синтаксис – совокупность правил некоторого языка, определяющих формирование его элементов. Синтаксис задается с помощью правил, которые описывают понятия некоторого языка. Примерами понятий являются – переменная, выражение, оператор, процедура.

Именно иерархия объектов, а не то, как они взаимодействуют между собой, определяются через синтаксис. Например, оператор может встречаться только в процедуре, а выражение в операторе, переменная может состоять из имени и необязательных индексов и так далее. Синтаксис не связан с такими явлениями в программе, как несоответствие типов или с тем, что переменная с данным именем не определена. Этим занимается семантика.

Семантика – правила и условия, определяющие соотношения между элементами языка и их смысловыми значениями, а также интерпретацию содержательного значения синтаксических конструкций языка.

Текст исходной программы на языке высокого уровня представляет собой обычный тестовый файл. Для его чтения и превращения в последовательность машинных команд, прежде всего, выполняется синтаксический анализ текста программы.

Синтаксический анализатор – компонента компилятора, осуществляющая проверку исходных операторов на соответствие синтаксическим правилам и семантике данного языка программирования.

Слайд 10

Процесс решения задач на компьютере – это совместная деятельность человека и компьютера.

На долю человека приходятся этапы, связанные с творческой деятельностью, – постановкой задачи, разработкой алгоритма вычислительного процесса, программированием задач и анализом результатов. На этом этапе участвует человек, хорошо представляющий предметную область задачи.

На долю персонального компьютера приходятся этапы обработки информации в соответствии с разработанным алгоритмом.

Языки программирования относятся к группе формальных языков, для которых в отличие от естественных языков однозначно определены синтаксис и семантика. Описание синтаксиса языка включает определение алфавита и правил построения различных конструкций языка из символов алфавита и более простых конструкций. Для этого обычно используют форму Бэ́куса – На́ура, сокращенно БНФ [бэ эн эф], синтаксические диаграммы.

Синтаксические диаграммы отображают правила построения конструкций в более наглядной форме.

Представлена синтаксическая диаграмма, иллюстрирующая первые два правила описания конструкции, – целое. Из диаграммы видно, что целое число может быть записано со знаком или без, и может включать произвольное количество цифр.

Расширенная форма Бэкуса – Наура, РБНФ [эр бэ эн эф] – формальная система определения синтаксиса, в которой одни синтаксические категории

последовательно определяются через другие. Используется она для описания контекстно-свободных формальных грамматик. Предложена Никла́усом Ви́ртом. Является расширенной переработкой форм Бэкуса – Наура, отличается от БНФ [бэ эн эф] более ёмкими конструкциями, позволяющими при той же выразительной способности упростить и сократить в объёме описание.

Слайд 11

Из символов алфавита в соответствии с правилами синтаксиса строят различные конструкции. Простейшей из них является конструкция – Идентификатор.

Конструкция Идентификатор используется во многих более сложных конструкциях для обозначения имен программных объектов, полей данных, процедур, функций и так далее.

Синтаксическая диаграмма идентификатора приведена на слайде. Семантику языка программирования закладывают в компилятор. Таким образом, синтаксически корректная программа, написанная на языке программирования, после преобразования ее в последовательность машинных команд обеспечит выполнение компьютером требуемых операций.

Такие же диаграммы существуют и для других объектов языка.

Слайд 12

Интегрированная среда разработки – комплексное средство, включающее всё необходимое для создания программного обеспечения. Интегрированная среда разработки – это комплекс программ для разработки, компиляции, ли́нкинга и отладки программ. На сегодняшний день

существует множество интегрированных сред для C++ [си плюс плюс] со своими особенностями, достоинствами и недостатками.

Ли́нкер – это программа, которая производит компоновку. Принимает на вход один или несколько объектных модулей и собирает по ним исполнимый модуль.

Перед выбором интегрированной среды необходимо определиться, для каких целей она нужна.

На слайде перечислены некоторые интегрированные среды для языка C++ [си плюс плюс].

«Visual Studio [вижуал сту́диэ] в основном известна для написания приложений. Это полный набор инструментов, позволяющий произвести точную отладку и настройку приложения. Есть как Community [кэммйу́ни ти]-версия, так и PRO[про].

Visual Studio [ви́жуал сту́диэ] предназначена не только для разработчиков на C++ [си плюс плюс], но также поддерживает многие другие популярные языки, такие как C# [си шарп], Visual Basic [вижуал бэ́йсик]. Visual Studio [вижуал студиэ], и предлагает множество функций:

- интеллектуальное автодополнение кода;
- дизайнер графических форм;
- простую в использовании навигационную систему.

Visual Studio [вижуал студиэ] может быть использован для разработки компьютерных программ для Microsoft Windows [ма́йкрософт ви́ндэус], а также веб-сайтов, веб-приложений и веб-сервисов» [1].

Слайд 13

«Codeblocks [код блокс] – это еще одна свободная и открытая среда для C++ [си плюс плюс]. Обратим внимание на некоторые из функций Codeblocks [код блокс]:

– простая и быстрая установка;
– наличие портативной версии;
– встроенная возможность создания блок-схем.

Dev-C++ [дэф си плюс плюс] – это бесплатная интегрированная среда разработки с открытым исходным ко́дом. Является довольно легкой средой, которой требуется всего пару минут для установки. Это лучшая среда разработки для новичков, поскольку в ней можно установить плагин для создания интерфейсов методом перетаскивания элементов. Некоторые из возможностей Dev-C++ [деф си плюс плюс]:

– малый вес;
– простая в использовании панель инструментов;
– автозавершение кода;
– простая установка.

CLion [си лайон] – платная программа, не имеющая бесплатной версии.

Возможности CLion [си лайон]:

– удобное создание визуализированных интерфейсов;
– наличие инструментов для удобного создания кода и отладки;
– возможность установки плагинов;
– поиск ошибок в коде в Live [лайф]-режиме» [3].

В первую очередь, выбор интегрированной среды зависит от уровня программирования. Для новичков лучшие среды разработки – это Dev-C++ [деф си плюс плюс] и Codeblocks [код блокс].

«Конечно, современные интегрированные среды разработки предлагают программистам гораздо больше возможностей, чем те, которые входят в описанный необходимый минимум. Например, многие современные интегрированные среды являются визуальными – они позволяют создавать интерфейс программы с помощью мышки, точно в таком виде, в каком он предстанет потом перед пользователем. Интегрированные среды, не

являющиеся визуальными, требуют от программиста писать специальный код, ответственный за создание пользовательского интерфейса программы» [3].

Слайд 14

Познакомимся с таким понятием, как сте́йтмент. Компьютерная программа – это последовательность инструкций, которые сообщают компьютеру, что ему нужно сделать.

«Сте́йтмент – это наиболее распространенный тип инструкций в программах. Это и есть та самая инструкция, наименьшая независимая единица в языке С ++ [си плюс плюс]. Сте́йтмент в программировании – это то же самое, что и предложение в русском языке. Мы пишем предложения, чтобы выразить какую-то идею. В языке С ++ [си плюс плюс] мы пишем сте́йтменты, чтобы выполнить какое-то задание. Все сте́йтменты в языке С ++ [си плюс плюс] заканчиваются точкой с запятой.

Есть много видов сте́йтментов в языке С++. Рассмотрим самые распространенные из них:

- сте́йтмент объявления – statement declaration [стэ́йтмент деклара́йшн]. Он сообщает компилятору имя и тип переменной. В программировании каждая переменная занимает определенное число адресуемых ячеек в памяти в зависимости от её типа. Минимальная адресуемая ячейка – байт. Все переменные в программе должны быть объявлены, прежде чем использованы;
- сте́йтмент присваивания – assignment statement [эсси́гмэнт стэ́йтмэнт];
- сте́йтмент вывода – output statement [а́утпут стэ́йтмент]. Мы выводим значение переменной на экран.

Компилятор может также обрабатывать выражения. Это математический объект, который создается для проведения вычислений и

нахождения результатов. Выражения могут содержать цифры, буквенные переменные, операции, математические функции.

В языке С ++ [си плюс плюс] сте́йтменты объединяются в блоки – функции. Функция – это последовательность сте́йтментов. Каждая программа, написанная на языке С ++ [си плюс плюс], должна содержать главную функцию main [мэйн]. Именно с первого сте́йтмента, находящегося в функции main [мэйн] и начинается выполнение всей программы. Функции, как правило, выполняют конкретное задание. Например, сте́йтменты определяют бо́льшее из заданных чисел или вычисляют среднюю оценку студентов по какой-либо дисциплине» [5].

Слайд 15

На слайде представлены критерии качества программы:

- «корректность. Очевидно, что программа должна работать правильно, иначе нет смысла ее писать;
- надежность. Программа не должна зависать или зацикливаться при любых исходных данных;
- эффективность. Программа должна использовать, по возможности, минимальное количество ресурсов памяти, хотя в настоящее время это стало менее актуальным.

А также:

- минимализа́ция времени работы программы. Эта проблема остается актуальной, особенно при обработке больших массивов данных;
- эргономичность – удобство для пользователя. Не стоит забывать о том, что первым пользователем вашей программы будете вы сами;
- переносимость. Программа должна работать не только на вашем компьютере, но и на других;

– читабельность – удобство для программиста. К сожалению, по прошествии времени программа забывается, и давно написанную программу невозможно прочитать как книгу. Приходится заново принимать все когда-то принятые решения. Поэтому нет смысла создавать себе дополнительные трудности в виде плохо структурированного и плохо читаемого текста.

Для решения этой проблемы существует ряд принципов написания текста программы;

– во-первых, не стоит писать, как слишком длинные строки – они уходят за пределы экрана по ширине, так и слишком короткие строки – они увеличивают длину не только всей программы, но и каждого отдельного блока, который вы в этом случае не можете охватить взглядом и оценить, что он делает;
– во-вторых, вложенные блоки принято писать со смещением вправо;
– в-третьих, не стоит прятать фигурные скобки в конец строки – в этом случае их трудно найти и определить начало и конец блока» [5].

Слайд 16

«Среди современных языков программирования язык C++ [си плюс плюс] является одним из наиболее распространенных. Это язык программирования общего назначения, цель которого сделать работу серьёзных программистов более приятным занятием. Язык C++ [си плюс плюс] обеспечивает гибкие и эффективные средства определения новых типов. Язык C++ [си плюс плюс] хорошо зарекомендовал себя эффективностью, лаконичностью записи алгоритмов, логической стройностью программ. Во многих случаях программы, написанные на языке C++ [си плюс плюс], сравнимы по скорости с программами, написанными на Ассéмблере, при этом они более наглядны и просты в сопровождении.

Перечислим некоторые особенности языка C++ [си плюс плюс]:

- в языке реализован ряд операций низкого уровня. Некоторые из таких операций напрямую соответствуют машинным командам, например, поразрядные операции или операции ++ [плюс плюс] и – – [минус минус];
- базовые типы данных языка отражают те же объекты, с которыми приходится иметь дело в программе на Ассе́мблере – байты, машинные слова и так далее» [5].

Слово «программа» происходит от двух греческих слов и в переводе означает «предписание», то есть, заданную последовательность действий.

Программа на алгоритмическом языке записывается с помощью символов, образующих алфавит языка. Он включает заглавные и прописные буквы латинского алфавита, арабские цифры и некоторые специальные символы.

Остальные символы могут быть использованы только в символьных строках, символьных константах и комментариях. На языке C++ [си плюс плюс] заглавные и прописные буквы различаются. Особенно это надо учитывать при составлении имен переменных, идентификаторов.

Слайд 17

Программа на языке C++ [си плюс плюс] состоит из директив препроцессора, описаний и определений, набора функций, среди которых обязательно есть одна с именем main [мэйн], и именно с нее начинается выполнение программы.

На слайде представлена простейшая программа на языке C++ [си плюс плюс]. Программа начинается с комментариев – первая строка программы. В комментариях можно использовать буквы русского алфавита.

Вторая строка содержит команду препроце́ссора. Препроце́ссор – это специальная программа, являющаяся частью компилятора языка C++ [си плюс плюс]. Она сообщает транслятору, что надо включить в программу описания, необходимые для работы стандартных потоков ввода-вывода, которые находятся в iostream [а́йострим].

Команда препроцессора include [и́нклюд] начинается с символа «решетка». Используя угловые скобки, мы сообщаем компилятору, что подключаемый заголовочный файл предоставляется Стандартной библиотекой.

Третья строка. Фраза, указанная в этой строке, означает – «…используя пространство имен std [эс ти дэй]».

«Запись using namespace std [ю́зинг нэ́ймспэйс эс ти дэй] говорит о том, что будет использоваться пространство имен std [эс ти дэй], в котором определена вся стандартная библиотека языка. Здесь пространство имен – это область действия объектов программы, в том числе переменных. В области имен std [эс ти дэй] описаны стандартные потоки ввода и вывода. Если не объявлять используемое пространство имен, нужно было бы явно указывать его при любом обращении к этим потокам» [3].

Четвертая строка – заголовок функции с именем main [мэйн]. Каждая программа на C++ [си плюс плюс] должна содержать только одну функцию с таким именем. Пустые круглые скобки – нужны для соблюдения синтаксиса обращения к функциям. Обычно в них помещают список формальных параметров. Пятая строка выдает приветствие – Привет всем.

Слайд 18

На слайде представлен фрагмент программы, содержащей функцию с именем summa [сумма].

Функция – это поименóванная часть программы, которая может вызываться из других частей программы столько раз, сколько необходимо. Первая часть определения функции задает ее имя, тип возвращаемого значения, если оно есть, а также типы и имена формальных параметров. Значение возвращается из функции с помощью оператора return [рит'óрн].

Тело любой функции, включая главную функцию main [мэйн], это последовательность описаний, определений, операторов, заключенных в фигурные скобки. Каждое описание, определение или оператор заканчивается символом точка с запятой. Переменные, описанные в теле любой функции, являются локальными и доступны в теле только данной функции. Переменные, которые описаны вне всяких функций, являются глобальными и доступны в каждой функции.

«Комментарии нужно писать так, чтобы человек, который не имеет ни малейшего представления о том, что делает ваш код, смог в нем разобраться. Однострочные комментарии – это комментарии, которые пишутся после символов // [два слеша]. Они пишутся в отдельных строках, и всё, что находится после этих символов, – комментирования, которые игнорируются компилятором. Как правило, однострочные комментарии используются для объяснения одной строчки кода» [1].

Многострочные комментарии начинаются с символов /* [слеш звездочка] и заканчиваются символами */ [звездочка слеш]. Еще раз напомним, что комментарии компилятором не обрабатываются.

Функция вывода информации на экран – cout [си аут].

Два рядом стоящих знака «меньше» – означают «поместить в выходной поток». В выходной поток помещается всё, что находится справа от этих символов. В данной программе – это строковая константа, заключённая в кавычки. Стро́ковая константа может содержать любые символы, включая и те, которые не изображаются на экране, а являются управляющими

символами, например – \n [слеш эн] – означают переход к началу следующей строки экрана.

Слайд 19

Препроцессор – это специальная программа, являющаяся частью компилятора языка С++ [си плюс плюс]. Она предназначена для предварительной обработки текста программы. Препроцессор позволяет включать в текст программы файлы. Работа препроцессора осуществляется с помощью специальных директив, указаний. Они отмечаются знаком «решетка».

Рассмотрим некоторые директивы препроцессора:

– include [и́нклюд] – вставляет текст из указанного файла, подключаются стандартные библиотеки, используются стандартные функции. На слайде приведены примеры подключения различных заголовочных файлов:
– iostream [айостри́м] – для работы с потоками ввода-вывода;
– fstream [эф стри́м] – для работы с файлами данных;
– math [мэс] – для использования математических функций;
– define [дэ фа́йн] – подстановка в тексте.

Директива include [инклю́д] позволяет включать в текст программы указанный файл. Если файл является стандартной библиотекой и находится в папке компилятора, он заключается в угловые скобки, если нет, он указывается в кавычках. Для файла, находящегося в другом каталоге необходимо в кавычках указать полный путь.

Директива define [дэ фа́йн] определяет подстановку в тексте программы. Она используется для определения символических констант. Указывается имя подстановки и текст подстановки. Все вхождения имени заменяются в коде на текст подстановки. В примере на слайде директива

define [дэ фа́йн] использована для ввода символической константы FILE [файл], связанной с именем файла f.dat [эф дат]. Указан и путь доступа к файлу. В дальнейшем в программе при обращении к файлу вместо имени файла используется символическая константа.

Имена символических констант рекомендуется записывать прописными буквами, чтобы зрительно отличать их от имен переменных и функций.

Слайд 20

Из символов языка формируются лексемы – это идентификаторы, ключевые слова, константы, знаки операций. Границы лексем определяются другими лексемами, такими как разделители или знаки операций.

Идентификаторы используются для описания имен переменных, имен функций, пользовательских типов данных – структур, констант и т. д.

Идентификатор – это имя программного объекта. В идентификаторе могут использоваться латинские буквы, цифры и знак подчеркивания.

Ключевые зарезервированные слова в языке С ++ [си плюс плюс] предназначены для специального использования. Ключевые слова могут быть использованы как имена стандартных библиотечных функций, как имена операторов, и не могут быть использованы в виде идентификаторов. На слайде приведены некоторые ключевые слова.

Далее мы подробно рассмотрим правила составления идентификаторов в программе.

Константами называют неизменяемые величины. Различаются целые, вещественные, символьные и строковые константы. Компилятор, выделив константу в качестве лексемы, относит ее к одному из типов по ее внешнему виду.

Знак операции – это один или более символов, определяющих действие над операндами. Выражение – это последовательность операндов, связанных знаками арифметических или логических операций. Операндом может быть переменная, числовая константа, стандартная функция. Внутри знака операции пробелы не допускаются. Операции делятся на унарные, бинарные и тернарную по количеству участвующих в них операндов. Большинство стандартных операций может быть переопределено, перегружено. Перегрузка операций будет рассмотрена далее.

Слайд 21

«В языке C++ [си плюс плюс] все идентификаторы, имена переменных, функций, классов и так далее, должны быть уникальными. Если в вашей программе находятся два одинаковых идентификатора, то будьте уверены, что ваша программа не скомпилируется, и вы получите ошибку конфликта имен.

Как только программы становятся больше, то и идентификаторов используется больше. Следовательно, вероятность возникновения конфликта имен значительно возрастает. Язык C ++ [си плюс плюс] предоставляет достаточно механизмов для предотвращения возникновения конфликтов имен, например, локальная область видимости или пространства имен. Самый простой способ сообщить компилятору, что определенный идентификатор находится в определенном пространстве имен, использовать оператор разрешения области видимости» [1].

Идентификаторы используются для описания имен переменных, функций, пользовательских типов данных – структур, констант и так далее.

На слайде приведены примеры имен переменных. Прописные и строчные буквы на языке C ++ [си плюс плюс] различаются.

Мы можем определять идентификаторы любыми словами, то есть любыми именами. Есть несколько общих правил, которые необходимо соблюдать:

- идентификатор не может быть ключевым словом. Ключевые слова зарезервированы;
- идентификатор может состоять только из букв нижнего или верхнего регистра, цифр или символов подчёркивания, все другие символы и пробелы запрещены;
- идентификатор должен начинаться с буквы, идентификатор не может начинаться с цифры или знака подчеркивания.

«Как правило, на языке С ++ [си плюс плюс] имена переменных и функций пишутся буквами в нижнем регистре. Имена идентификаторов, которые начинаются с заглавной буквы, используются как имена структур, классов или перечислений, о чем мы поговорим позже.

Если имя переменной или функции состоит из нескольких слов, то есть два варианта: разделить подчёркиванием или использовать принцип написания нескольких слов слитно, без пробелов. И каждое новое слово начинать с заглавной буквы – CamelCase [кэ́мэл кейс]. Что в переводе означает «верблюжий стиль». Он получил своё название из-за заглавных букв, которые напоминают верблюжьи горбы» [1].

На слайде приведены некоторые ключевые слова. Язык С++ [си плюс плюс] имеет зарезервированный набор из 84 [восьмидесяти четырех] слов для собственного использования, даже в версии С++ 17 [си плюс плюс семнадцать]. Эти слова называются ключевыми словами, каждое из которых имеет свое особое значение. В ходе изучения нашего курса мы будем знакомиться с ключевыми словами и их назначением.

Слайд 22

Литера́лы – это ограниченная последовательность символов алфавита языка, представляющая собой изображение фиксированного, неизменяемого объекта.

В программе можно использовать литера́лы, которые могут быть следующих типов:

- целые;
- вещественные, с фиксированной или с плавающей точкой;
- символьные;
- стро́ковые.

Целые литералы могут быть десятичные, восьмери́чные и шестнадцатери́чные. На слайде приведены примеры литералов, констант различных типов.

Десятичные литералы – это, последовательность цифр со знаком плюс или минус.

Восьмеричные литералы начинаются с литеры ноль, а шестнадцатеричные – с двух литерал, ноль и икс, причем икс может быть прописной или заглавной.

В зависимости от значения литерала компилятор присваивает ей тот или иной тип, а именно – char [чар], int [инт] или long int [ло́нг инт].

Литералы можно использовать в арифметических выражениях. Помимо неименованных констант, в выражениях можно использовать и именованные литералы.

Слайд 23

Вещественные литералы бывают с фиксированной или с плавающей точкой.

Числа с плавающей точкой, называемые также числами с плавающей запятой, – это вещественные числа, в которых число хранится в виде мантиссы и порядка, то есть показателя степени. При этом число с плавающей точкой имеет фиксированную относительную точность и изменяющуюся абсолютную.

Реализация математических операций с числами с плавающей точкой в вычислительных системах может быть как аппаратной, так и программной.

Число с плавающей точкой состоит из следующих частей:

- знака мантиссы, который указывает на отрицательность или положительность числа;
- мантиссы, выражающей значение числа без учёта порядка;
- знака порядка;
- порядка, то есть выражения степени основания числа, на которое умножается мантисса.

По умолчанию компилятор присваивает вещественному числу тип double [дабл]. В вещественных литералах используется десятичная точка, разделяющая целую и дробную части.

Слайд 24

Символьные литералы – это один символ, например – z [зэт], как показано на слайде.

В качестве символьных литералов также могут использоваться управляющие коды, не имеющие графического представления. При этом код управляющего символа начинается с символа «обратный слеш». На слайде приведены некоторые управляющие символы.

«Все символьные литералы имеют тип char [чар] и занимают в памяти один байт. Значением символьной константы является числовое значение её

внутреннего кода. Коды символов представлены в международной таблице ASCII [а́ски] кодов.

На языке C++ [си плюс плюс] допустимы и стро́ковые литералы. Строковые литералы представляют собой последовательность любых символов, заключенных в двойные кавычки. Кавычки не входят в строку, а лишь ограничивают её.

Технически строковые литералы представляют собой массив символов, и по этому признаку могут быть отнесены к разряду сложных объектов языка» [4].

Строковые литералы мы подробно рассмотрим в следующих темах нашего курса.

Слайд 25

«Объем памяти, который использует переменная, зависит от типа данных этой переменной. Так как мы, как правило, получаем доступ к памяти через имена переменных, а не через адреса памяти, то компилятор может скрывать от нас все детали работы с переменными разных размеров.

Есть несколько причин, по которым полезно знать, какой объем памяти занимает определенная переменная, определенного типа.

Компьютеры имеют ограниченное количество свободной памяти. Каждый раз, когда мы объявляем переменную, небольшая часть этой свободной памяти выделяется до тех пор, пока переменная существует. Фактический размер переменных может отличаться на разных компьютерах, поэтому для его определения используют операцию sizeof [са́йз оф]» [3].

Все переменные, используемые в программе на языке C++ [си плюс плюс], должны быть обязательно описаны. При описании переменных указывается имя переменной и тип, определяющий объем памяти,

выделяемый под хранение переменной. Описывать переменные можно в любом месте программы, но до первого обращения к этой переменной.

В таблице на слайде представлены основные типы данных языка C++ [си плюс плюс].

Логический тип bool [бул] используется исключительно для хранения результатов логических выражений, которые могут принимать одно из двух значений true [тру], что означает «истина», или false [фолс], что означает «ложь». Этот тип переменных часто называют бу́левыми переменными.

Константе true [тру] эквивалентны все числа от одного до двухсот пятидесяти пяти включительно, тогда как константе false [фолс] эквивалентно только одно целое число – ноль.

Тип данных char [чар] – это символьный тип данных, который используется для представления символов. То есть, каждому символу соответствует определённое двоичное число из диапазона от нуля (0) до двухсот пятидесяти пяти. Для представления символов в C++ [си плюс плюс] типу данных char [чар] отводится один байт, в одном байте – восемь бит, с помощью которых можно закодировать 256 символов. Коды символов представлены в таблице ASCII [а́ски] кодов.

Типы float [флэт] и double [дабл] используются для представления вещественных чисел.

Слайд 26

«Диапазон целочи́сленной переменной определяется двумя факторами: её размером, который измеряется в байтах, и её знаком, который может быть signed [сайнд] или unsigned [ан са́йнт].

Целочисленный тип signed [сайнт], со знаком, означает, что переменная может содержать как положительные, так и отрицательные числа.

Целочисленный тип unsigned [ан са́йнт], без знака, может содержать только положительные числа.

Если попытаться использовать значение, которое находится вне диапазона значений определенного типа данных, то произойдет переполнение. Переполнение случается из-за того, что переменной не было выделено достаточно памяти для их хранения» [4].

На слайде приведены типы данных для представления целых чисел**.** Они характеризуются определенным размером занимаемой памяти в байтах и диапазоном принимаемых значений. Напоминаем, в зависимости от компилятора размер занимаемой памяти и диапазон принимаемых значений могут изменяться.

Диапазон принимаемых значений меняется в случае, если тип данных объявляется с приставкой unsigned [ан са́йнт] – без знака. Приставка unsigned [ан са́йнт] говорит о том, что тип данных не может хранить знаковые значения, и диапазон положительных значений увеличивается в два раза.

Преобразования типов нужны в C++ [си плюс плюс] потому, что арифметические операции со смешанными типами являются нормой для языков, используемых в числовых задачах.

«Синтаксически тип void [войд] является фундаментальным типом. Его можно использовать только как часть сложного типа, так как объектов типа void [войд] не существует. Этот тип используют для указания на то, что функция не возвращает значения. Подробнее этот тип мы рассмотрим в следующих темах» [5].

Слайд 27

Тема 2. Операции и стандартные библиотечные функции языка

Одни и те же знаки операций могут использоваться в различных выражениях и по-разному интерпретируются в зависимости от контекста.

Знаки операций делятся на унáрные и бинáрные. Унáрными называются такие операции, в которых участвует один операнд. Операндом может быть переменная, константа, функция.

В таблице приведены унáрные операции с примерами их применения.

Операция получения адреса операнда позволяет ввести переменную целого типа, в которой запоминается адрес операнда. Эта переменная называется переменная-указатель.

Операция обращения по адресу – доступ по адресу к значению того объекта, на который указывает операнд, что дает доступ к такой информации как через имя переменной, так и непосредственно по адресу, где расположена данная информация.

Слайд 28

Операция «побúтовое отрицание» означает поразрядное инвертирование внутреннего двоичного кода целочисленного аргумента.

Операция логического отрицания только в одном случае дает значение «единица», что означает «истину», если аргумент равен нулю. Во всех остальных случаях результат «ноль» означает «ложь».

Инкремéнт и декремéнт – это операции увеличения на единицу и уменьшение на единицу. Применяются только для левостоя́щих операндов. Операндом может быть только переменная.

Эти операции бывают:

- префиксными – увеличение или уменьшение значения операнда до его первого использования;
- постфиксными – увеличение или уменьшение значения операнда после его использования.

На слайде приведены примеры операции инкремент. В зависимости от записи этой операции результаты вычислений различны. Во втором примере

эта операция указана в скобках. Правила записи арифметических операций не позволяют в выражениях ставить подряд два знака. По этой причине знак декреме́нт, увеличение на единицу, указан в скобках.

Слайд 29

На данном слайде – продолжение таблицы уна́рных операций.

Функцию sizeof [са́йз оф] можно использовать в двух вариантах, когда аргументом могут быть любое арифметическое выражение или стандартные типы данных.

С помощью операции sizeof [са́йз оф] можно определить объем памяти любого выражения или объем памяти, отводимой под переменные определенного типа. Аргументом функции является обозначение стандартных типов данных. Это очень удобно, так как в разных системах объем памяти, предусмотренный для хранения переменных разных типов данных, может быть различным.

Применение операции sizeof [са́йз оф] к массиву дает количество байтов, занимаемых массивом, а не количество его элементов и не размер в байтах каждого из них.

Применение операции sizeof [са́йз оф] к указателю дает размер самого указателя, а не объекта, на который он указывает.

Слайд 30

«Арифметические выражения в любом языке программирования записываются в строчку. Они могут содержать константы, то есть постоянные значения, имена переменных, знаки арифметических операций, круглые скобки, которые нужны для изменения порядка действий, и вызовы функций. При определении порядка действий используется приоритет, так называемое старшинство операций. Если в выражение входят переменные

различных типов, в некоторых случаях происходит автоматическое приведение типа к более широкому. Например, результат умножения целого числа на вещественное – это вещественное число. Результат деления целого числа на целое – это всегда целое число, остаток при делении отбрасывается. Когда нужно получить вещественный результат деления, одно из чисел, делимое или делитель, нужно преобразовать в вещественный тип – для числа поставить десятичную точку, а для переменной или выражения использовать явное приведение типа» [6].

В бинарных операциях участвуют два опера́нда. Бинарные операции делятся на аддити́вные и мультипликати́вные.

К аддити́вным операциям относятся операции – бинарный плюс, бинарный минус и умножение операндов.

При выполнении аддитивных операций выполняется основное правило – приоритет выполнения операций. На слайде приведены выражения, в которых используются аддитивные операции. Изменить приоритет выполнения операций можно с помощью скобок. Если в выражении встречаются операции, имеющие одинаковый приоритет, то в силу вступает правило их выполнения слева направо.

При использовании аддитивных операций операндами могут быть любые выражения.

Эти операции являются операциями присваивания, поэтому слева от знака равенства допустимы только имена переменных, а справа – любое арифметическое выражение.

Слайд 31

К мультипликативным операциям относятся деление операндов и получение остатка от деления целочисленных операндов.

«Операции умножения и деления выполняются над целыми и вещественными операндами. Типы первого и второго операндов могут отличаться. При этом выполняются преобразования операндов по умолчанию. Типом результата является тип операндов после преобразования.

В процессе выполнения мультипликативных операций ситуация переполнения или потери значимости не контролируется. Если результат мультипликативной операции не может быть представлен типом операндов после преобразования, то информация теряется.

Операция деления выполняет деление первого своего операнда на второй. Если оба операнда являются целыми значениями и не делятся нацело, то результат округляется в сторону нуля. Деление на ноль дает ошибку во время выполнения» [4].

При делении целочисленных операндов результат округляется до целого. Знак результата будет отрицательным, если отрицательным является или делимое или делитель.

При выполнении операции остаток от деления целочисленных операндов – знак результата зависит только от знака делимого.

Особое внимание следует уделить результатам выполнения этих операций при отрицательных операндах. На слайде приведены примеры, которые демонстрируют указанные правила.

Слайд 32

«В далеком прошлом объем компьютерной памяти было очень мал, и ею очень дорожили. Это было стимулом максимально разумно использовать каждый доступный бит. Используя поби́товые операторы, можно создавать функции, которые позволят уместить восемь значений логического типа в переменную размером один байт, что значительно сэкономит потребление памяти. В прошлом такой трюк был очень популярен.

В настоящее время объем памяти стал существенно больше, и программисты обнаружили, что лучше писать код так, чтобы было проще и понятнее его поддерживать, нежели усложнять его ради незначительной экономии памяти. Поэтому спрос на использование побитовых операторов несколько уменьшился, за исключением случаев, когда необходима максимальная оптимизация» [5].

Рассмотрим операции сдвига. Операция «сдвиг влево» битового представления значения левого целочисленного операнда на количество разрядов правого целочисленного операнда на **n** [эн] позиции соответствует увеличению первого операнда в **2n** [два эн] раза.

«Операция «сдвиг вправо» битового представления значения левого целочисленного операнда на количество разрядов правого целочисленного операнда соответствует уменьшению первого операнда в **2n** [два эн] раза с отбрасыванием дробной части результата.

Операции битового сдвига могут быть полезны при декодировании информации от внешних устройств и для чтения информации о статусе. Операторы битового сдвига могут также использоваться для выполнения быстрого умножения и деления целых чисел. Сдвиг влево равносилен умножению на 2 [два], а сдвиг вправо – делению на 2 [два], как показано в примере в таблице» [4].

Приведенные примеры демонстрируют работу операций сдвига с указанием двоичного представления чисел и результатов выполнения этих операций.

Слайд 33

На данном слайде приведены поразрядные логические операции.

Конъю́нкция – это поразрядная логическая операция, приближенная к союзу «И». Синоним этого действия – логическое умножение. В этом случае

сравнивается каждый бит первого операнда с соответствующим битом второго операнда. Если оба бита равны единице, то и результат равен единице, в ином случае он равен нулю.

Дизъю́нкция – это поразрядная логическая операция, по своему применению максимально приближённая к союзу «ИЛИ» – в смысле «или то, или это, или оба сразу». То есть это сложение, при котором сравнивается каждый бит первого операнда с соответствующим битом второго операнда. Если хотя бы один или оба бита равны единице, то результат равен единице, в ином случае он равен нулю.

При выполнении поразрядной логической операция исключающего «или» сравнивается каждый бит первого операнда с соответствующим битом второго. Если один бит равен нулю, а второй – единице, в результате устанавливается единица. Если оба бита равны нулю или оба равны единице, то результат будет равен нулю.

Приведенные примеры демонстрируют работу поразрядных операций.

Слайд 34

В программах на языке С++ [си плюс плюс] операции отношений используются для сравнения двух величин между собой. Этими величинами могут быть числа, переменные, константы, результаты вычислений выражений. Операции отношений возвращают одно из двух возможных значений:

- true [тру], если результат операции отношения исполняется, то есть «истинно»;
- False [фолз], если результат операции отношения не исполняется, то «есть ложно».

Операции отношений являются бина́рными. Они требуют двух операндов.

В операциях отношения операнды должны быть арифметического типа, результат всегда целочисленный.

Обратите внимание на операцию «равно». Это два рядом стоящих знака равенства. Теперь обратите внимание на знак «не равно». Это восклицательный знак и знак равенства.

Операции отношений могут использоваться:

- в циклах, где есть условие выполнения цикла;
- в операторах присваивания, содержащих логические выражения, если нужно определить результат сложного логического выражения;
- в операторах условного перехода if [иф].

Если проверяется одновременно несколько условий, то они должны быть связаны операциями отношений.

Слайд 35

В логических операциях, операнды должны быть арифметического типа, результат всегда целочисленный.

В логических операциях результатом может быть одно из двух значений:

- ноль, что означает «ложь»;
- единица, что означает «истина».

Логические операции «и» и «или» используются как логические связки, если одновременно проверяется несколько условий в операторах проверки условий.

Результатом операции «И» будет «истина», если истинны все входящие в выражение операнды.

Результатом операции «ИЛИ» будет «истина», если истинным является хотя бы один из операндов, входящих в выражение.

На слайде приведены примеры логических операций.

Слайд 36

«Присваивание значений переменных делается командой присваивания. На языке С++ [си плюс плюс] имеются помимо привычных и сокращенные формы записи арифметических операций» [7].

На слайде во втором столбце приведены примеры сокращенной формы записи арифметических операций присваивания на С++ [си плюс плюс] и в третьем столбце – привычная запись.

В операциях присваивания участвует левый операнд и выражение, стоящее справа от знака равенства.

На данном слайде показаны два варианта использования операций присваивания. Однако, считается хорошим тоном использовать на языке С++ [си плюс плюс] запись операций присваивания, как показано во второй колонке данной таблицы.

В выражении, стоящим справа от знака равенства, участвует операнд, стоящий слева от знака равенства.

При выполнении арифметических операций соблюдается иерархия выполнения арифметических операций.

Изменить приоритет выполнения арифметических операций можно с помощью скобок. Если в выражении участвуют операции, имеющие одинаковый приоритет, в силу вступает правило их выполнения слева направо.

«Обратите внимание на первый пример. Допустимы смешанные арифметические выражения, то есть в одном арифметическом выражении присутствуют операнды различных типов.

Наличие хотя бы одного вещественного операнда приводит к получению результата вещественного. При делении, если в операции участвуют два целых числа, то результат деления будет округляться до

целого числа, даже если результат присваивается переменной вещественной переменной. Чтобы результат представлял число с плавающей точкой, один из операндов также должен представлять число с плавающей точкой, то есть применить явное преобразование типов» [7].

В С++ [си плюс плюс] можно совмещать операции присваивания. Обратите внимание на второй пример.

Слайд 37

Программа может вызывать множество функций из реализации стандартной библиотеки С++ [си плюс плюс]. Данные функции выполняют важные задачи, например занимаются вводом и выводом, а также предоставляют эффективные реализации часто используемых операций.

На слайде представлена таблица стандартных математических функций. Для использования стандартных функций необходимо подключить библиотеку стандартных функций, заголовочный файл «Математика». В первой строке таблицы показано название стандартной библиотеки, которую необходимо подключить к программе для использования этих функций.

В таблице указаны тип аргумента, для которого можно использовать функции, и тип получаемого результата. То есть эти функции используются для вещественных аргументов.

Не забывайте об этом условии использования стандартных библиотечных функций, иначе могут возникнуть сообщения об ошибках при компиляции программы.

Аргумент любой функции указывается в скобках, он может быть любым арифметическим выражением. Не забывайте о том, что тип арифметического выражения зависит от типов входящих в выражение операндов. Аргументы тригонометрических функций должны быть указаны в радианах. Если значение аргумента функции имеет значение в градусах, его

необходимо перевести в радианы. На слайде приведена формула перевода из градусов в радианы – формула 1 [один].

Слайд 38

На языке С++ [си плюс плюс] нет ключевых слов, обеспечивающих ввод–вывод, обрабатывающих строки, выполняющих различные математические вычисления или какие-нибудь другие полезные процедуры. Все эти операции выполняются за счет использования набора библиотечных функций, поддерживаемых компилятором. Существуют два основных вида библиотек – библиотека С [си], которая поддерживается всеми компиляторами и библиотека классов С++ [си плюс плюс], которая годится только для языка С++ [си плюс плюс]. Прежде чем программа сможет использовать какую-нибудь библиотеку функций, она должна включить соответствующий заголовок. Под заголовками понимают заголовочные файлы.

На слайде приведены стандартные функции, использование которых требует подключения стандартной библиотеки «Математика».

Рассмотрим функцию вычисления модуля числа. В зависимости от типа аргумента, целого типа или вещественного типа, используются разные функции.

Аргументом этих функций могут быть любые арифметические выражения. Аргумент указывается справа от имени функции в скобках.

При выполнении арифметических операций у стандартных функций наивысший приоритет выполнения перед арифметическими операциями.

При использовании этих функций, необходимо следить за типом аргументов перечисленных функций. В таблице указаны типы аргументов и тип результата.

Слайд 39

При вычислении арифметических операций соблюдаются следующие правила:

- соблюдается приоритет выполнения арифметических операций;
- два знака не могут следовать один за другим;
- если в выражении есть знаки одинакового приоритета, в силу вступает правило их выполнения слева направо;
- изменить приоритет выполнения арифметических операций можно с помощью скобок.

Приоритетом называется очередность выполнения операций в выражении. Операция, находящаяся между двумя операциями с равными приоритетами, связывается с той операцией, которая находится слева. Выражение, заключенное в скобки, перед выполнением, вычисляется как отдельный операнд.

Любое выражение языка состоит из операндов, переменных, констант, соединенных знаками операций. Операции выполняются в строгой последовательности. Величина, определяющая преимущественное право на выполнение той или иной операции, называется приоритетом. Порядок выполнения операций может регулироваться с помощью круглых скобок.

Слайд 40

В верхней строке таблицы приведено название заголовочного файла, который необходимо подключить к программе при обращении к функциям, приведенным в таблице.

Функция floor [флор] вычисляет наименьшее ближайшее целое число, не превышающее x [икс]. Обратите внимание на пример применения этой функции к отрицательным числам.

Следует обратить внимание на результат работы этой функции для отрицательных аргументов. В таблице приведены примеры.

Функция fmod [эф мод] вычисляет остаток от деления x [икс] на y [игрек]. Результат применения этой функции имеет вещественный тип.

Разберем подробнее пример, приведенный в последней строке таблицы. Аргументом функции вычисления корня квадратного является выражение, состоящее из двух слагаемых, соответственно, все выражение заключено в скобки. Знаменатель, состоящий из двух слагаемых, также заключается в скобки. Иначе выражение исказится.

Слайд 41

На языке C++ [си плюс плюс] допустимы смешанные выражения, то есть выражения, в которые входят операнды разных типов.

«При выполнении бинарных операций производятся преобразования по умолчанию для приведения операндов к одному и тому же типу, который потом используется как тип результата:

- если один из операндов имеет тип double [дабл], другой тоже преобразуется в double [дабл];
- если один операнд имеет тип float [флэ́ут], то второй операнд преобразуется к типу float [флэ́ут];
- bool [боул] преобразуется в int [инт];
- если один операнд имеет тип long int [ло́нг инт], то второй операнд преобразуется к типу long int [лонг инт];
- если один операнд имеет тип unsigned int [ан са́йнт инт], то второй операнд преобразуется к типу unsigned int [ан са́йнт инт].

В языке C++ [си плюс плюс] нет операций преобразования между символом и кодом символа, так как в оперативной памяти символ итак

хранится в виде его кода. Поэтому можно к переменной, хранящей символ, прибавить единицу и получить следующий символ.

Операция приведения типа продемонстрирована на слайде. При преобразовании типов надо помнить о том, что при преобразовании между знаковыми или беззнаковыми значениями и при преобразовании от типа с большей размерностью к типу с меньшей размерностью могут возникнуть ошибки» [1].

Слайд 42

Для закрепления пройденного материала на слайде приведены некоторые операции.

В таблице поэтапно приведены операции и наименования операций.

При начальном значении переменной, равном пяти, выполните указанные операции.

При выполнении операций сдвига, конъюнкции и дизъюнкции необходимо использовать двоичное представление чисел. Сдвиг влево соответствует увеличению левого операнда, справа доставляются нули в количестве, равном правому операнду в выражении. Сдвиг вправо означает уменьшение левого операнда. Удаляются разряды справа на количество, равное правому операнду в выражении.

Если полученный вами результат равен четырем, это означает что вы правильно выполнили указанные операции.

Слайд 43

Тема 3.1. Базовые алгоритмические структуры (Часть 1)

При решении прикладных задач с использованием компьютера можно выделить ряд последовательных этапов.

На первом этапе выполняется постановка задачи, где четко определяются исходные данные и накладываемые на них ограничения и конечные результаты.

За постановкой задачи следует этап формализации. На этом этапе строится модель решения задачи. Создавая модель, нужно выделить существенные стороны рассматриваемого процесса или явления, определить математические соотношения, связывающие результаты решения задачи с исходными данными.

На основе модели строится алгоритм решения задачи – последовательность шагов от ввода исходных данных до вывода результатов.

Этап программирования предполагает выбор языка программирования, запись алгоритма на выбранном языке.

Программа должна пройти этапы отладки и тестирования. Под отладкой программы понимается процесс испытания работы программы и исправления обнаруженных ошибок. Ошибки могут быть синтаксическими, вызванными нарушением правил записи языка программирования, или логическими, когда нет соответствия последовательности действий в программе последовательности действий в разработанном алгоритме. Синтаксические ошибки обнаруживаются системой программирования. Логические ошибки выявляются на этапе тестирования.

Тест – это конкретный набор значений исходных данных, для которого известен результат решения задачи, используемый для проверки правильности работы программы.

Этап решения задачи – это выполнение расчетов по программе для получения результатов решения задачи.

Последний этап – это анализ получаемых результатов, иногда может потребовать вернуться к одному из предшествующих этапов, чтобы внести изменения в модель или алгоритм, или программу.

Слайд 44

Современные системы программирования на C++ [си плюс плюс] состоят из нескольких составных частей. Это такие части, как сама среда программирования, язык, Стандартная библиотека С-функций [си-функций] и различные библиотеки С-классов [си-классов]. Как правило, чтобы выполнить программу, необходимо пройти через шесть этапов, перечисленных на слайде.

Первый этап представляет создание и редактирование файла с исходным текстом программы. Он может выполняться с помощью простейшего редактора текстов программ или с помощью встроенного редактора интегрированной среды.

На втором этапе компилятор начинает препроцессорную обработку текста программы, прежде чем ее компилировать. Следует знать, что в системе C++ [си плюс плюс] программирования перед началом этапа самой трансляции всегда выполняется программа предварительной обработки. Она отыскивает директивы трансляции, или директивы препроцессора. Директивы препроцессора указывают на то, какие нужно выполнить преобразования перед трансляцией исходного текста программы. Обычно это включение других текстовых файлов в файл, который подлежит компиляции. Препроцессорная обработка инициируется компилятором перед тем, как программа будет преобразована в машинный код.

Третий этап – это компиляция. Как правило, программы на языке C++ [си плюс плюс] содержат ссылки на различные функции, которые определены вне самой программы. Например, в стандартных библиотеках или в личных библиотеках программистов.

Четвертый этап – компоновка. Компоновщик связывает объектный код с кодами отсутствующих функций и создает, таким образом, исполняемый загрузочный модуль.

Пятый этап – загрузка. Перед выполнением программа должна быть размещена в памяти. Это делается с помощью загрузчика, который забирает загрузочный модуль программы с диска и перемещает его в память.

Шестой этап – это выполнение. Каждый из названных этапов может заканчиваться ошибкой. Тогда программист должен вернуться к редактированию исходного текста программы. Затем снова пройти через все этапы работы с исходным текстом программы до получения работающего без ошибок загрузочного модуля.

Слайд 45

Рассмотрим подробнее один из ключевых этапов решения задач – разработку алгоритмов.

Эффективность работы алгоритма и правильность полученных результатов обеспечивается следующими свойствами алгоритма:

- понятность – алгоритм должен включать однозначные предписания доступные исполнителю, в роли которого может быть человек, робот или компьютер;
- дискретность – представление алгоритма в виде последовательности элементарных шагов;
- массовость – применимость алгоритма к некоторому множеству исходных данных;
- определенность – за конечное число шагов алгоритм должен приводить либо к решению поставленной задачи, либо к остановке с выдачей сообщения о невозможности получить решение;

– однозначность – повторное применение алгоритма к одним и тем же исходным данным должно приводить к получению одного и того же результата.

Если разработанный алгоритм не удовлетворяет какому-либо из названных свойств, то в дальнейшем это приведет к нежелательным последствиям, которые могут быть выявлены на одном из следующих этапов работы с задачей. Так невыполнение свойства массовости алгоритма приведет к тому, что он окажется применимым только для обработки одного заданного набора исходных данных, а для других данных придется разрабатывать новый алгоритм.

Например, можно разработать алгоритм вычисления площади треугольника с заданными конкретными длинами сторон. Этот алгоритм не будет удовлетворять условию массовости. Нужен алгоритм, позволяющий определить площадь любого треугольника. Конкретные значения длин сторон треугольника должны задаваться при выполнении алгоритма.

Слайд 46

Любой прибор, купленный в магазине, снабжается инструкцией по его использованию. Данная инструкция и является алгоритмом для правильной эксплуатации прибора.

Каждый шофёр должен знать правила дорожного движения. Правила дорожного движения однозначно регламентируют поведение каждого участника движения. Зная эти правила, шофер должен действовать по определенному алгоритму.

Массовый выпуск автомобилей стал возможен только тогда, когда был придуман порядок сборки машины на конвейере. Определенный порядок сборки автомобилей – это набор действий, в результате которых получается автомобиль.

Используются следующие формы представления алгоритмов:

- словесная – запись алгоритма на естественном разговорном языке;
- графическая – использование для представления алгоритма графических фигур, когда каждому действию ставится в соответствие определенная фигура;
- псевдокод – описание алгоритма на условном алгоритмическом языке, включающем как элементы, характерные для языка программирования, так и фразы разговорного языка, а также общепринятые математические и иные обозначения;
- программа – запись алгоритма на одном из языков программирования.

Слайд 47

В блок-схеме каждому типу действий, например, вводу исходных данных, выводу результата или проверке некоторого условия соответствует блок определенной формы. Форма блока однозначно определяет вид выполняемого действия.

Блоки соединяются линиями переходов, определяющими очередность выполнения действий. Блок-схема выстраивается в направлении либо сверху вниз, либо слева направо. В каждый блок может входить сколько угодно соединительных линий, определяющих переход к данному действию, кроме блока – начало алгоритма. Из каждого блока может быть только один выход, чтобы обеспечить свойство однозначности алгоритма. Только блок проверки условия может иметь два выхода, позволяющие реализовать разветвление процесса вычислений.

Блоки, с названиями начало и конец, используются для обозначения начала или конца алгоритма. Блок начала всегда один, а блоков, обозначающих конец алгоритма, может быть несколько. Однако, желательно, использовать и один блок конца алгоритма.

Блок ввода данных предполагает ручной ввод значений переменных. Имена переменных, которые необходимо ввести, указываются внутри этого блока.

Блок присваивания значений используется для вычисления значений по формулам и присваивания вычисленных значений переменным.

Блок проверки условий предполагает выбор одного из двух альтернативных путей работы алгоритма в зависимости от выполнения или невыполнения проверяемого условия. Условие записывается внутри блока.

Блок для организации цикла обеспечивает многократное выполнение некоторой последовательности действий, которая называется телом цикла. Этот блок применяется в тех алгоритмах, где количество повторений цикла можно определить заранее. Внутри блока указывается переменная, являющаяся параметром цикла, задается ее начальное значение, конечное значение и шаг изменения.

Блок вывода результатов предполагает вывод значений переменных, имена которых указываются внутри блока.

Слайд 48

«Из перечисленных блоков составляют структуры алгоритмов. Структурами называют ограниченный набор блоков и стандартных способов их соединения для выполнения типичных последовательностей действий. Чаще других используется графическое представление алгоритмов, при котором алгоритм изображается в виде последовательности связанных между собой геометрических фигур, называемых функциональными блоками. Такое графическое представление называется схемой алгоритма или блок-схемой.

Алгоритмы собирают из трех основных базовых структур – следование, ветвление, цикл. Однако в качестве элементов этой структуры могут выступать и базовые структуры. Именно поэтому правильнее называть

элементы такой структуры функциональными блоками, и в дальнейшем под функциональными блоками будем понимать не только блок «действие», но и любую базовую структуру» [7].

Построению последовательности действий решения задачи предшествует определение типа вычислительного процесса. Выделяют линейные, разветвляющиеся и циклические алгоритмические структуры, которые можно считать базовыми.

Основной характеристикой базовых алгоритмических структур является наличие в них одного входа и одного выхода.

Алгоритмическая структура, в которой естественный ход выполнения действий не нарушается, и действия выполняются строго в том порядке, как они записаны, называется «следование». На слайде представлена алгоритмическая структура типа «следование».

Под действиями предполагается применение таких операций, как ввод значений исходных данных, присваивание переменным каких-либо значений, вычисление значений переменных по заданным выражениям, вывод полученных результатов.

Слайд 49

В алгоритмах, содержащих базовую структуру типа «ветвление», естественный ход вычислений может быть нарушен в зависимости от проверки какого-либо условия.

Выбирается один из возможных путей прохождения алгоритма. Независимо от того, какой путь будет выбран, выполнение алгоритма будет продолжаться, но всегда может быть выбран только один из возможных путей. Это обеспечивает свойство однозначности алгоритма.

Разветвляющиеся алгоритмы существуют в двух основных вариантах. Алгоритмические структуры типа ветвление показаны на слайде.

При использовании конфигурации «если–то» проверяется условие, и если оно выполняется, то выполняется указанное действие. Если условие не выполняется, действие игнорируется. Такая структура называется «неполной развилкой». Блок, обозначающий «действие», может содержать любую последовательность операций.

При использовании конфигурации типа «если–то – иначе» также проверяется условие, и если оно выполняется, то выполняется действие 1 [один], иначе выполняется действие 2 [два]. Такая структура называется «полной развилкой».

Выбор альтернативного пути дальнейшего прохождения алгоритма осуществляется указанием на линиях выхода из блока проверки условия слов «да» или «нет».

Слайд 50

Структура цикла состоит из логического элемента с проверкой условия и функционального блока, называемого «телом цикла». Ясно, что тело цикла может выполняться неоднократно. Базовая алгоритмическая структура типа цикл предполагает многократное выполнение некоторой совокупности действий.

Различают циклические структуры трех типов:

- цикл с параметром;
- цикл с предусловием;
- цикл с постусловием.

Для организации любого цикла необходимо выполнить три действия:

- определить начальное значение параметра цикла;
- сформировать текущее значение параметра цикла;
- осуществить проверку на выход из цикла.

«В тех случаях, когда заранее известно число повторений цикла, используют цикл с параметром так называемый цикл «ДЛЯ». Этот блок позволяет сделать схему алгоритма более компактной и наглядной. В свою очередь, частным случаем цикла «ДЛЯ» является цикл «повторять n раз» или цикл со счетчиком, который очень часто используется в вычислительных задачах. В языках программирования, как правило, такой цикл организуется при помощи специального оператора» [7].

В алгоритмической структуре типа «Цикл с параметром», представленной на слайде, указанные действия выполняются в блоке организации цикла. Здесь i [ай] – параметр цикла, a [эй] – начальное значение параметра цикла, b [би] – конечное значение параметра цикла, d [ди] – шаг изменения параметра цикла.

Действия, которые повторяются многократно, образуют тело цикла. Цикл с параметром используют в том случае, когда количество повторений цикла заранее известно.

Слайд 51

Циклы, для которых число повторений цикла заранее определить невозможно, называются итерационными. Базовые алгоритмические структуры для организации итерационных циклов существуют в двух вариантах – цикл с предусловием и цикл с постусловием. Выполнение некоторого условия определяет прекращение повторений цикла.

В вычислительных алгоритмах циклу с предусловием выполнения тела цикла предшествует проверка условия продолжения цикла.

Ряд действий, образующих тело цикла, повторяется до тех пор, пока указанное условие выполняется. Если условие не выполнилось с первого раза, цикл не будет пройден ни разу.

В вычислительных алгоритмах типа цикла с постусловием сначала выполняется ряд действий, образующих тело цикла, а затем проверяется условие продолжения цикла. Цикл повторяется до тех пор, пока не выполнится указанное условие. Если условие не выполнилось с первого раза, цикл все равно будет пройден один раз.

Алгоритмические структуры цикл с предусловием и цикл с постусловием можно использовать и в случае известного заранее числа повторений цикла.

Тело цикла для всех трёх разновидностей алгоритмических структур может содержать внутри себя любую последовательность действий, включая структуру типа ветвление и даже цикл.

Мы рассмотрели средства представления алгоритмов с помощью блок-схем. Далее будут показаны примеры составления блок-схем алгоритмов для решения некоторых типовых вычислительных задач.

Слайд 52

На слайде представлен алгоритм решения следующей задачи.

Ежемесячный размер премии сотрудника предприятия составляет 35 % от оклада. Подоходный налог, взимаемый с сотрудника, составляет 13 %. Требуется вычислить сумму к выдаче сотруднику с учетом премии и подоходного налога.

На этапе постановки задачи определяются исходные данные. В данной задаче к исходным данным относится только размер оклада сотрудника.

Результатом вычислений будет являться сумма к выдаче.

Подоходный налог взимается со всей суммы, причитающейся сотруднику, которая складывается из оклада и премии.

Используемые обозначения:

ОК [оу кей] – оклад сотрудника;

P [пи] – размер премии сотрудника;

PN [пи эн] – подоходный налог;

S [эс] – сумма к выдаче.

Перед составлением алгоритма необходимо определить тип вычислительного процесса. Анализ алгоритма показывает, что он не содержит структур ветвления или циклов, значит, тип вычислительного процесса и реализующего его алгоритма – линейный.

Последовательность действий, описанных в блок-схеме:

- ввод значения переменной OK [оу кей] – размер оклада сотрудника;
- вычисление размера премии сотрудника P [пи];
- вычисление подоходного налога PN [пи эн] с учетом оклада и премии;
- подсчет суммы к выдаче S [эс], которая складывается из суммы оклада сотрудника и премии за вычетом подоходного налога;
- вывод результата.

Слайд 53

Рассмотрим еще один пример алгоритма линейного типа.

Условие задачи. Рабочий в течение месяца изготавливал на своем станке два вида деталей. На изготовление одной единицы первого вида деталей он затрачивал A [а] часов, на изготовление второго вида деталей – затрачивал B [бэ] часов рабочего времени. Определить количество деталей второго вида, изготовленных рабочим за месяц, если известно количество изготовленных в этом месяце деталей первого вида и общее количество часов, отработанных рабочим в течение месяца.

Для данной задачи исходными данными являются количество деталей первого вида, изготовленные рабочим за месяц, и общее количество часов, отработанные рабочим в этом месяце. Время на изготовление одной детали

каждого вида также должно быть задано. Результатом вычислений является количество деталей второго вида.

На слайде представлен алгоритм решения задачи.

Используемые обозначения:

K1 [ка один] – количество деталей первого вида;

K2 [ка два] – количество деталей второго вида;

A [а] – время, затраченное на изготовление одной детали первого вида;

B [бэ] – время, затраченное на изготовление одной детали второго вида.

А также:

S [эс] – общее количество часов, отработанных рабочим в этом месяце;

S1 [эс один] – количество часов, затраченное на изготовление всех деталей первого вида;

S2 [эс два] – количество часов, затраченное на изготовление всех деталей второго вида.

Суммарное количество часов, затраченное рабочим на изготовление деталей первого вида, вычисляется как произведение количества деталей первого вида на время, затраченное на одну деталь.

Количество часов, затраченное на изготовление деталей второго вида, вычисляется как разница общего количества часов, отработанных в этом месяце, и количеством часов, затраченных на изготовление деталей первого вида.

Зная время изготовления одной детали, можно подсчитать количество деталей второго вида.

Анализ последовательности действий позволяет определить тип вычислительного процесса как следование.

Слайд 54

Перейдем к рассмотрению алгоритмов разветвленного типа. Отметим, что каждый такой алгоритм должен содержать как минимум одну базовую алгоритмическую структуру – ветвление.

На слайде представлен алгоритм решения следующей задачи.

Большее из двух произвольных заданных чисел следует заменить средним арифметическим значением этих чисел, а в случае их равенства оставить числа без изменения.

Исходными данными для этой задачи являются два произвольных числа, результатом вычислений – преобразованные в соответствии с условием задачи значения этих чисел.

В результате выполнения алгоритма большее из двух чисел заменится их средним арифметическим значением, меньшее – не изменится. В случае равенства чисел алгоритм выберет ветвь по направлению, обозначенному словом «нет», второе число заменится средним двух чисел. А так как числа равны, то среднее значение будет равно самому числу. Следовательно, числа останутся без изменения.

Используемые обозначения:

X [икс] – первое число;

Y [игрек] – второе число.

Перед составлением алгоритма определим тип вычислительного процесса. Необходимо проверить условие и выбрать один из альтернативных путей решения. Это означает, что алгоритм будет построен с использованием базовой структуры типа ветвление.

Последовательность действий, описанных в блок-схеме:

– ввод значений двух переменных;

– проверка условия и выбор дальнейшего пути вычислений: если первое число больше второго, первое заменить средним значением этих чисел, в противном случае – заменить второе;

– вывод результата.

При любых значениях введенных чисел алгоритм пройдет по одному из возможных путей и приведет к выводу результата.

Слайд 55

Рассмотрим еще один пример алгоритма типа ветвление.

На слайде представлен алгоритм решения следующей задачи.

Плата за пользование услугами сети Интернет некоторого провайдера осуществляется по входящему трафику. Схема оплаты: от 0 до 100 Мбайт [ста мегабайт] – фиксированная плата, сверх 100 Мбайт [ста мегабайт] – взимается дополнительная плата за каждый Мбайт [мегабайт] входящего трафика. Составить алгоритм расчета оплаты услуг провайдера, если известен объем входящего трафика, стоимость базового трафика объемом 100 Мбайт [сто мегабайт] и стоимость 1 Мбайта [одного мегабайта] сверх базового трафика. При этом за не использованный полностью базовый трафик провайдер пересчета оплаты услуг не производит.

Используемые обозначения:

A [эй] – объем входящего трафика;

P [пи] – размер платы за первые 100 Мбайт [мегабайт], базовый трафик;

DP [ди пи] – стоимость одного Мбайт [мегабайта], после превышения базового трафика;

S [эс] – размер платы за пользование услугами Интернет.

Оплата услуг провайдера зависит от объема входящего трафика. Если величина трафика не превышает 100 Мбайт [ста мегабайт], то оплата

фиксированная, иначе складывается из стоимости базового трафика и доплаты за превышение трафика. Произведение количества Мбайт [мегабайт], превышающих базовый, и стоимости одного мегабайта определяет размер доплаты.

Анализ последовательности действий, требующих проверки условия для выбора альтернативного пути вычислений, позволяет определить тип вычислительного процесса как ветвление.

Слайд 56

Рассмотрим задачу. Найти сумму первых **N** [эн] натуральных чисел.

Напомним, что натуральными называются целые положительные числа 1, 2, 3 [один два три] и так далее.

На слайде представлены два варианта решения этой задачи.

Вариант 1

Представлена базовая структура типа цикл с предусловием. Блоки пронумерованы для удобства описания комментариев к алгоритму.

Блок 1 [один] – определение начального значения параметра цикла.

Блок 2 [два] – проверка условия выхода из цикла.

Блок 3 [три] – вычисление суммы чисел.

Блок 4 [четыре] – формирование текущего значения параметра цикла.

Блок 5 [пять] – вывод результата.

Действия, указанные в блоках 3 и 4, образуют тело цикла.

Действия, указанные в блоках 1, 2 и 4, необходимы для организации цикла.

Вариант 2

Количество повторений цикла в данной задаче заранее известно, оно равно N [эн]. Это позволяет построить алгоритм с использованием базовой структуры – цикл с параметром.

В блоке 1 указан закон изменения параметра цикла i [ай], определено начальное значение равное единице, конечное значение, равное N [эн] и шаг изменения параметра цикла, равный единице.

В блоке 2 подсчитывается сумма чисел.

В блоке 3 выполняется вывод результата.

Действие, указанное в блоке 2, образует тело цикла.

В блоке 1 объединены действия, указанные в блоках 1, 2 и 4 блок-схемы из варианта 1.

Второй вариант блок-схемы более компактен, что наглядно демонстрирует рациональность применения цикла с параметром в задачах с известным заранее количеством повторений.

Слайд 57

Рассмотрим следующую задачу.

Задана последовательность целых чисел. Каждое последующее число отличается от предыдущего на единицу. Найти сумму положительных и сумму отрицательных чисел этой последовательности, если известен диапазон изменения этих чисел.

Поясним, что, например, в диапазон целых чисел от минус двух до плюс трёх, входит два отрицательных числа – минус два и минус один, три положительных числа – один, два и три, а также – число ноль. Тогда сумма отрицательных чисел указанного диапазона составит минус три, сумма положительных – плюс шесть. Число ноль, будучи добавленным к любой из сумм, не изменит её.

Для решения этой задачи исходными данными являются начало и конец диапазона чисел. Алгоритм задачи должен формировать целые числа из заданного диапазона, учитывая, что каждое последующее число вычисляется из предыдущего добавлением к нему единицы. Каждое число

анализируется и добавляется либо к сумме положительных, либо к сумме отрицательных чисел.

Используемые обозначения:

a [эй] – начало диапазона целых чисел, первое число;

b [би] – конец диапазона целых чисел, последнее число;

i [ай] – число из заданного диапазона;

SP [эс пи] – сумма положительных чисел;

SO [эс оу] – сумма отрицательных чисел.

Алгоритм решения задачи показан на слайде.

Анализ последовательности действий позволяет определить тип вычислительного процесса как цикл. Переменную i [ай] можно использовать как параметр цикла, значение которого меняется от a [эй] до b [би] с шагом равным единице, и построить базовую алгоритмическую структуру цикл с параметром.

Однако в теле цикла необходимо анализировать каждое число, является оно положительным или отрицательным. То есть в теле цикла присутствует элемент ветвления.

Последовательность действий, описанных в блок-схеме:

- ввод исходных данных – начало и конец диапазона целых чисел;
- построение цикла с использованием блока цикла, в котором указан закон изменения параметра цикла i [ай] – от a до b с шагом, равным 1 [единице];
- проверка условия – если число отрицательное, то выбирается ветвь, обозначенная словом «да», иначе – ветвь обозначенная словом «нет»;
- вывод результатов – сумма положительных чисел SP [эс пи] и сумма отрицательных чисел SO [эс оу] в заданном диапазоне.

Слайд 58

На слайде представлена структура вложенных циклов.

Вложенные циклы строятся по тем же правилам, что и простые циклы.

Цикл, охватывающий все циклы, называется внешним, а циклы входящие во внешний цикл, называются внутренними.

При построении вложенных циклов необходимо помнить о следующем правиле – параметр внешнего цикла меняется медленнее, чем параметр внутреннего цикла. При одном значении параметра внешнего цикла, параметр внутреннего пробегает все свои возможные значения.

На слайде представлен фрагмент блок-схемы, демонстрирующий вывод на печать таблицы умножения.

Параметром внешнего цикла является переменная j [джей]. Параметром внутреннего цикла является переменная i [ай].

«Первый шаг в написании программы – записать на естественном языке, возможно, с применением обобщенных блок-схем, что именно и как она должна делать. Если вы не можете сформулировать алгоритм по-русски, велика вероятность того, что он плохо продуман. Такое описание алгоритма полезно по нескольким причинам. Оно помогает в деталях продумать алгоритм, найти на самой ранней стадии некоторые ошибки, разбить программу на логическую последовательность блоков, а также обеспечить комментарии к программе.

После написания программу следует тщательно обработать – убрать ненужные фрагменты, сгруппировать описания, оптимизировать проверки условий и циклы, проверить, оптимально ли разбиение на функции и так далее» [3].

Слайд 59

Программа представляет собой последовательность инструкций и операторов.

Каждый элемент языка определяется синтаксисом программирования. Это элемент языка, задающий полное описание действия, которое необходимо выполнить. Каждый оператор представляет собой законченную фразу языка программирования и определяет некоторый вполне законченный этап обработки данных. В состав операторов могут входить служебные слова, данные, выражения и другие операторы. Каждый оператор в любом языке программирования имеет определенный синтаксис. Под синтаксисом оператора понимается система правил, грамматика, определяющая его запись с помощью элементов алфавита данного языка, в который наряду с различными символами входят, например, и служебные слова.

Пустой оператор состоит из знака точки с запятой, перед которым нет никаких выражений, по нему не выполняется никаких действий. Он используется там, где синтаксис языка требует присутствие оператора, а по смыслу никакие действия и не должны выполняться.

Составной оператор – это последовательность операторов, заключенная в фигурные скобки. После фигурной скобки точка с запятой не ставится.

На слайде приведен пример составного оператора.

Если среди операторов в фигурных скобках, имеются определение и описание, то составной оператор превращается в блок, где все определенные в нем переменные становятся локальными и доступными только в пределах данного блока.

Синтаксически и составной оператор, и блок являются отдельными операторами. Внутри блока или составного оператора все операторы заканчиваются символом точка с запятой.

Операторы бывают исполняемые и неисполняемые. Исполняемые операторы задают действия над данными. Неисполняемые операторы служат для описания данных, поэтому их зачастую называют операторами описания или просто описаниями.

Слайд 60

«Каждый программный объект имеет область действия, которая определяется видом и местом его объявления. Существуют следующие области действия – блок, файл, функция, прототип функции и поименованная область.

Идентификаторы, описанные внутри блока, являются локальными. Область действия идентификатора начинается в точке определения и заканчивается в конце блока, видимость – в пределах блока и внутренних блоков, время жизни – до выхода из блока. После выхода из блока память освобождается.

Идентификаторы, описанные вне любого блока, функции или пространства имен, имеют глобальную видимость и постоянное время жизни и могут использоваться с момента их определения.

Единственными идентификаторами, имеющими такую область действия, являются метки операторов. В одной функции все метки должны различаться, но могут совпадать с метками других функций» [3].

Идентификаторы, указанные в списке параметров прототипа функции, имеют областью действия только прототип функции. С понятием «прототип функции» мы познакомимся в последующих темах.

«Поименованная область. C++ [си плюс плюс] позволяет явным образом задать область определения имен как часть глобальной области. Область видимости совпадает с областью действия за исключением ситуации, когда во вложенном блоке описана переменная с таким же именем.

В этом случае внешняя переменная во вложенном блоке невидима, хотя он и входит в ее область действия. Способ обратиться к скрытой локальной переменной отсутствует.

В каждой области действия различают так называемые пространства имен. Пространство имен – область, в пределах которой идентификатор должен быть уникальным. В разных пространствах имена могут совпадать, поскольку разрешение ссылок осуществляется по контексту идентификатора в программе» [3].

Слайд 61

На языке C++ [си плюс плюс] нет встроенных средств ввода–вывода, нет отдельных операторов ввода–вывода.

На языке C++ [си плюс плюс] разработана библиотека ввода–вывода iostream [айостри́м], которая определяет стандартные потоки ввода–вывода:

- cin [син] – стандартный входной поток;
- cout [си а́ут] – стандартный выходной поток.

Для выполнения операций ввода–вывода переопределены две операции:

- получить из входного потока;
- поместить в выходной поток.

На слайде приведены фрагменты ввода–вывода информации.

При вводе информации из входного потока читается последовательность символов до пробела, затем эта последовательность преобразуется к типу идентификатора, и получаемое значение помещается в идентификатор. Допускается ввод нескольких значений идентификаторов одновременно. При этом одно значение от другого отделяется пробелом или нажатием клавиши Enter.

При выводе информации на печать значение преобразуется в последовательность символов и выводится в выходной поток. При выводе значений нескольких переменных используют управляющие символы:

– \n [обратный слеш эн] – для перевода курсора на следующую строку;
– \t [обратный слеш ти] – для табуляции;
– endl [э́ндэл] – манипулятор для перевода курсора на следующую строку.

Слайд 62

Как уже отмечалось, каждый элемент языка определяется синтаксисом и семантикой. Синтаксические определения устанавливают правила построения элементов языка, а семантика определяет их смысл и правила использования. Объединенная единым алгоритмом совокупность описаний и операторов образует программу на алгоритмическом языке.

На данном слайде рассмотрена задача вычисления значения функции по заданной формуле – формула 2 [два]. Перед решением любой задачи необходимо определить множество исходных данных и ожидаемый результат.

Значения исходных данных a [эй] и x [икс] вводятся с клавиатуры, результат вычислений y [игрек] выводится на печать. В программу желательно включать вывод на печать комментарий, сопровождающий вычисления . Например, значения каких переменных необходимо ввести с клавиатуры, как показано в приведенном примере. Вывод результата также сопровождается комментариями, значение какой переменной выводится на экран.

Тип вычислительного процесса в данной задаче – базовая алгоритмическая структура «следование». Действия выполняются строго в том порядке, в каком записаны.

Ввод информации с клавиатуры и вывод результатов сопровождаются комментариями, выводимыми на экран. В строке комментариев при выводе результата, использован управляющий символ /n [слеш эн] – перевод курсора на начало следующей строки. Такие же действия выполняет и функция endl [э́ндэл].

С помощью препроцессорной команды include [инклю́д] подключены библиотеки:

- для организации ввода–вывода;
- для использования математической функции синус.

Слайд 63

Отдельные операторы, программы могут иметь метки. Метка – это уникальный идентификатор. Переход по метке осуществляется с помощью оператора goto [гоу ту]. Описания и определения также являются операторами, и перед ними также можно ставить метки. Метки могут быть символьными и числовыми. Символьная метка – это уникальный идентификатор. Числовые метки находятся в диапазоне от единицы до числа 9999 [девять тысяч девятьсот девяносто девять].

Оператор goto [гоу ту] уже давно не рекомендуют использовать, так как не существует таких ситуаций, где он был бы единственным решением. На языке типа C++ [си плюс плюс], имеющем богатый набор структур управления и предоставляющем дополнительные элементы управления типа break [брэйк] и continue [конти́ние], в нем нет необходимости. Перечисленные операторы будут рассмотрены далее.

Перед оператором, к которому осуществляется переход, ставится метка и знак двоеточие. Переходить по метке можно как выше, так и ниже по программе. Нельзя переходить к действиям в теле цикла.

Метка должна находиться в той же функции, что и сам оператор goto [гоу ту].

Слайд 64

Тема 3.2. Базовые алгоритмические структуры (Часть 2)

Возникают ситуации, когда в зависимости от каких-либо условий, появляется необходимость выполнения различных действий.

Например, если у студента по всем экзаменам оценка «отлично», он получает повышенную стипендию. Если есть хотя бы одна оценка – «неудовлетворительно», студент лишается стипендии, и так далее.

Для решения подобных задач используют конструкции ветвления.

На слайде показана структура условного оператора if [иф]. Часть оператора, указанная в квадратных скобках, является не обязательной, может быть опущена.

Если указанное выражение истинно, то есть не равно нулю, то выполняется оператор p1 [пи 1], если ложно, то выполняется оператор p2 [пи 2]. В качестве операторов p1 [пи 1] и p2 [пи 2] нельзя использовать описание или определение. Условие обязательно ставится в скобках.

Однако p1 [пи 1] и p2 [пи 2] могут быть простыми или составными операторами или блоками. Составной оператор указывается в фигурных скобках.

Слайд 65

На слайде приведены примеры использования условного оператора.

В первом примере, если указанное условие не выполняется, то есть ложно, выполняется составной оператор, блок. Так как внутри этого составного оператора присутствует описание переменной, он является блоком.

«В программе на языке C++ [си плюс плюс] идентификаторы используются для ссылок на различного рода объекты – функции, переменные, формальные параметры и тому подобное. При соблюдении определенных правил допускается использование одного и того же идентификатора для более чем одного программного объекта.

Чтобы различать идентификаторы объектов различного рода, компилятор языка устанавливает так называемые пространства имен. Во избежание противоречий имена внутри одного пространства должны быть уникальными. Однако в различных пространствах могут содержаться идентичные имена. Это означает, что можно использовать один и тот же идентификатор для двух или более различных объектов, если имена объектов принадлежат к различным пространствам. Однозначное разрешение вопроса о том, на какой объект ссылается идентификатор, компилятор языка осуществляет по контексту появления данного идентификатора в программе.

Уникальность имен в пределах этого пространства тесно связана с понятием области действия. Это выражается в том, что в данном пространстве могут содержаться совпадающие идентификаторы, если области действия именуемых ими объектов не пересекаются. Другими словами, совпадение идентификаторов возможно только при локальном переобъявлении. Поэтому переобъявление формальных параметров внутри любого из блоков функции недопустимо» [4].

Переменная i [ай], описанная в начале фрагмента, является глобальной, переменная i [ай] описанная внутри составного оператора является локальной и доступна только внутри этого составного оператора. Две переменные с одинаковыми именами находятся в разных ячейках памяти и имеют различные значения. Точка с запятой после фигурной скобки нужна обязательно, так как является концом оператора.

Во втором примере используется усеченная форма оператора if [иф]. Если условие не выполняется, то есть ложно, управление передается следующему оператору.

Слайд 66

В первом примере с помощью фигурных скобок определена последовательность выполнения оператора. Подчеркнутое слово else [элс] относится к первому оператору if [иф].

Во втором примере показано, что в условном операторе можно проверять одновременно несколько условий. Вся последовательность логических выражений указывается в фигурных скобках. Это обусловлено синтаксисом языка C++ [си плюс плюс].

При проверке нескольких условий в структуре логического оператора if [иф], их необходимо связать знаками логических отношений.

Во втором примере используется логическая связка «И». Все указанное условие будет истинным тогда и только тогда, когда оба указанных логических выражения истинны.

В третьем примере указана логическая связка «ИЛИ», то есть все указанное выражение будет истинно, если истинно хотя бы одно из указанных логических выражений.

Слайд 67

На слайде приведен фрагмент программы определения номера четверти, которой принадлежит точка с координатами x [икс] и y [игрек].

Задано дополнительное условие: x [икс] умноженное на y [игрек] не равно нулю, то есть точка не лежит на осях координат.

На рисунке расписаны сочетания значений координат в различных координатных четвертях.

Для решения этой задачи использована вложенная структура логического оператора if [иф]. Вся задача фактически реализована в одном операторе.

При использовании вложенных логических операторов в силу вступает правило: структура else [элс] относится к ближайшему оператору if [иф].

В приведенном фрагменте во вложенной структуре используется три оператора if [иф]. Для наглядности операторы подчеркнуты линиями разных цветов.

Изменить последовательность выполнения вложенных операторов можно с помощью фигурных скобок.

«Для записи каждого фрагмента алгоритма необходимо использовать наиболее подходящие средства языка. Любой цикл можно, в принципе, реализовать с помощью операторов goto [гоу ту] и if [иф], но это было бы нелепо, поскольку с помощью операторов цикла те же действия легче читаются, а компилятор генерирует более эффективный код. Ветвление на несколько направлений предпочтительнее реализовывать с помощью оператора switch [свич], а не нескольких операторов if [иф]» [5].

Слайд 68

«Во избежание большой вложенности операторов if [иф] используют оператор выбора switch [свич]. На слайде показана структура оператора.

Переключатель начинается с заголовка, определяющего имя метки. Тело переключателя заключено в фигурные скобки. Текст тела переключателя разделен метками case [кейс]. Двоеточие – признак метки» [7].

В структуре оператора указано выражение – это любое арифметическое выражение. По этому оператору сначала вычисляется указанное выражение. В частном случае выражением может быть одна переменная, значение

которой и анализируется. Выражение – это арифметическое выражение, где тип результата вычисления указанного выражения зависит от входящих в него операндов, констант, переменных и даже функций.

Если значение выражения равно n1 [эн один], то выполняется последовательность операторов P1, P2, …Pn [пи один, пи два и так далее, до пи энное].

Если значение выражения равно n2 [эн два], то выполняется последовательность операторов P2, P4, …Pn [пи два, пи четыри и так далее, до пи энное]. В противном случае – выполняется оператор p [пи]. В квадратных скобках указана не обязательная часть оператора.

На языке C++ [си плюс плюс] есть возможность прервать выполнение всей цепочки операторов. Оператор прерывания будет рассмотрен далее.

Слайд 69

На слайде показаны примеры использования оператора выбора. Иногда его называют переключателем.

Если возникает необходимость реализации определенного оператора, константа которого равна текущему значению указанного выражения, а не всей цепочки операторов, приходится прерывать оператор.

Чтобы прервать оператор switch [свич] используют оператор break [брэйк], который не допускает дальнейшего перебора и передает управление следующему оператору. Это продемонстрировано в первом примере.

Если значение переменной c [цэ] будет равно единице, на экране появится сообщение – «понедельник», и управление перейдет к следующему оператору в программе. Если значение переменной c [цэ] будет равно двойке, на экране появится сообщение – «вторник», и управление перейдет к следующему оператору в программе. Если значение с [цэ] не будет равно ни

одному из перечисленных значений, появится сообщение – «день не определен».

Во втором примере показана возможность перечисления значений в операторе switch [свич]. Это оптимизирует запись этого оператора.

Слайд 70

На слайде предложена задача вычисления значения функции y [игрек] в зависимости от введенного значения переменной x [икс] по формуле 3 [три]. Возможны три варианта решения этой задачи, вычисления значения функции:

- при x [икс] равном нулю;
- при x [икс] в диапазоне от нуля до 90 градусов;
- при x [икс] меньше нуля или больше 90 градусов.

При любом значении x [икс] алгоритм должен выбрать только один вариант решения. Тип вычислительного процесса в данной задаче – ветвление.

Реализуя свойство результативности алгоритма, в случае, когда значение аргумента не попадает в указанный диапазон, необходимо вывести сообщение о том, что функция не определена.

Значение аргумента задано в градусах. Напомним, стандартные тригонометрические функции воспринимают аргумент в радианах, поэтому в программе необходимо будет предусмотреть перевод аргумента из градусов в радианы.

Слайд 71

На слайде представлен первый фрагмент программы.

Подключены три заголовочных файла:

- для реализации ввода–вывода;

– для использования математических функций;
– для работы функции getch [гетч].

Функция getch [гетч] приостанавливает выполнение программы, пока не будет нажата клавиша ENTER [энтэр]. Эта функция используется, как правило, последней в программе, чтобы проанализировать полученные результаты.

Все переменные, используемые в программе, должны быть описаны до первого обращения к ним. Ввод сопровождается выводом на экран сообщения.

Исходными данными являются аргумент x [икс] и коэффициент a [а], значения которых вводятся с клавиатуры. Переменная y [игрек] введена для вычисления значения функции.

Слайд 72

Встречаются ситуации, когда программе нужно выбрать, какую операцию ей выполнить в зависимости от определенного условия. Во втором фрагменте программы анализируется значение аргумента функции в градусах, и в зависимости от его значения выбирается дальнейший путь вычислений.

В программе используется структура вложенных операторов if [иф].

Вспомним структуру логического оператора. С помощью фигурных скобок построен составной оператор, состоящий из двух операторов, действий:

– выбор формулы вычисления значения функции в зависимости от значения аргумента;
– вывод результата на экран.

Напомним, что аргументы тригонометрических функций воспринимаются компилятором в радианах. Поэтому непосредственно при

обращении к стандартной функции вставлена формула перевода значения аргумента из градусов в радианы.

Вывод результата сопровождается комментариями.

Слайд 73

Рассмотрим задачу: задана точка с координатами x [икс] и y [игрек] и радиус некоторого круга R [эр].

Определить, где находится точка с указанными координатами: на круге, внутри круга или вне круга с заданным радиусом. Дополнительное условие, указанное на слайде, означает, что точка не лежит на осях координат.

На слайде представлен фрагмент программы.

Для решения этой задачи необходимо определить радиус круга, на котором лежит точка с заданными координатами. Радиус круга подсчитывается по теореме Пифагора.

Сравнив радиус заданного круга R [эр] с радиусом круга, на котором лежит точка R1 [эр один], можно будет определить, где расположена заданная точка.

Тип вычислительного процесса – ветвление.

Слайд 74

Существует три оператора цикла:

- с предусловием;
- с постусловием;
- с параметром.

На данном слайде представлена структура оператора цикла с предусловием while [вайл].

Тело цикла – это простой или составной оператор, любая последовательность операторов, допустимых на языке C++ [си плюс плюс], заключенная в фигурные скобки. Если тело цикла состоит из одного оператора, фигурные скобки можно опустить. Если в теле цикла присутствует описание переменных, составной оператор является блоком. Все переменные, описанные в блоке, являются локальными и доступными, то есть видимыми только внутри этого блока.

В соответствии с этим оператором цикл повторяется до тех пор, пока указанное условие не станет равным нулю, то есть станет соответствовать условию «истинно». Если условие не выполнилось с первого раза, цикл ни разу не будет пройден.

Слайд 75

Рассмотрим задачу вычисления значений функции на заданном интервале. На слайде представлена функция – формула 4.

Исходными данными в этой задаче являются значения границ интервала и количество точек, которые необходимо ввести с клавиатуры.

Требуется найти максимальное значение функции на заданном интервале. Для этого введем дополнительную переменную, которой присвоим значение функции в первой точке. Это левая граница интервала.

На слайде представлена формула 5 – вычисление шага изменения аргумента. Подсчет шага изменения аргумента по указанной формуле позволит ровно за n [эн] шагов дойти от точки a [а] до точки b [бэ].

Тип вычислительного процесса – цикл.

Для решения этой задачи будем использовать оператор цикла с предусловием.

Слайд 76

На слайде приведена программа указанной задачи.

Подключены заголовочные файлы для реализации ввода, вывода и использования математических функций.

Введены следующие переменные:

a, b [а и б] – границы интервала;

n [эн] – количество точек;

y, x [икс и игрек] – значения функции и аргумента.

А также:

dx [де икс] – шаг изменения аргумента;

max [макс] – максимальное значение функции на заданном интервале.

Так как по условию задачи значения функции необходимо вычислить в n [эн] равностоящих точках, шаг изменения аргумента вычисляется по формуле, приведенной в программе.

Слайд 77

Для организации цикла необходимо выполнить три действия:

- определить начальное значение параметра цикла;
- сформировать текущее значение параметра цикла;
- выполнить проверку на выход из цикла.

Отсутствие любого из этих действий не позволит правильно организовать цикл.

Все эти действия реализованы в нашей программе. Для четкости понимания эти действия подчеркнуты на слайде.

При каждом прохождении через цикл вычисляется значение функции, печатается и осуществляется поиск максимального значения. Первоначально значению максимального значения присвоено значение функции в первой точке, в левой границе интервала.

Цикл продолжается до тех пор, пока значение аргумента не переберет все свои значения на заданном интервале.

Слайд 78

На слайде приведена структура оператора с постусловием.

Оператор do while [ду вайл] выполняется до тех пор, пока выполняется указанное условие, то есть пока условие «истинно», не равно нулю. Тело цикла заключается в фигурные скобки. Телом цикла может быть любая последовательность операторов, описаний языка С++ [си плюс плюс].

Если тело цикла – простой оператор, то фигурные скобки можно опустить.

В соответствии с данным оператором, если указанное условие не выполнилось с первого раза, цикл все равно будет пройден один раз. Для организации цикла необходимо определить параметр цикла, значение которого необходимо изменять в теле цикла.

В приведенном на слайде примере параметром цикла является переменная i [ай].

Слайд 79

На слайде демонстрируется задача вычисления суммы ряда с заданной точностью – формула 6. Количество циклов заранее определить невозможно. Процесс вычисления будет прекращен тогда, когда будет достигнута заданная точность. Считается, что точность достигнута, если очередной член ряда по модулю станет меньше заданной точности.

Циклические вычислительные процессы, в которых заранее невозможно определить количество циклов, называются итерационными циклическими вычислительными процессами.

Приведена формула 7 – формула вычисления общего члена ряда.

Для оптимизации процесса вычислений можно каждый последующий член ряда вычислить через предыдущий. Ряд знакопеременный. Изменения знака каждого члена ряда будем определять через формулу 8 – через числитель. Каждый числитель последующего члена ряда вычисляется через числитель предыдущего члена ряда с обратным знаком.

Можно добавить еще один внутренний цикл для вычисления значения факториала на каждом шаге итерации, то есть значение знаменателя для каждого члена ряда. Однако можно знаменатель каждого последующего члена ряда вычислить через знаменатель предыдущего члена ряда, доумножая на значение параметра цикла, – формула 9.

Слайд 80

На слайде приведена программа вычисления суммы ряда. В данной программе используется функция очистки экрана – clearscreen [клиа скрин] или cls [си эл эс].

Переменная i [ай] служит параметром цикла, на каждом шаге итерации значение которой увеличивается на единицу. Обратите внимание на начальные значения, присвоенные переменным – для вычисления числителя и для вычисления знаменателя. Эти значения подобраны так, чтобы по указанным формулам сформировался первый член ряда. Значение переменной s [эс] необходимо обнулить. Как известно, на языке С ++ [си плюс плюс] значения переменных после их описания автоматически не обнуляются.

Условием выхода из цикла является проверка, является ли очередной член ряда по модулю меньше заданной точности эпсилон. Проверка идет по модулю, так как ряд знакопеременный. С увеличением значения параметра цикла числитель увеличивается медленнее, чем знаменатель, то есть с увеличением i [ай] дробь уменьшается. Если на очередной итерации

величина члена ряда становится настолько мала, что ее добавление к сумме ряда уже не меняет, допустим, пятый знак после запятой, считается, что заданная точность достигнута.

В данной программе используется оператор цикла с постусловием.

После окончания цикла, распечатав значение переменной i [ай], можно определить количество итераций, циклов, за которые была достигнута заданная точность.

Слайд 81

На слайде приведена структура оператора цикла с параметром.

Оператор цикла с параметром имеет следующие элементы:

- инициализация;
- условие;
- выражение.

Эти элементы разделяются знаком «точка с запятой».

Тело цикла – любая последовательность операторов, заключенных в фигурные скобки. Если тело цикла – простой оператор, фигурные скобки можно опустить.

По этому оператору выполняется инициализация, проверяется указанное условии и, если оно истинно, вычисляется тело цикла. Инициализация – это описание и определение начальных значений некоторых переменных.

Необходимо помнить, что если условие не выполнилось с первого раза, цикл ни разу не будет пройден.

Инициализация выполняется только один раз – при входе в оператор цикла. Проверяется условие, и если оно истинно, выполняется тело цикла.

После каждого цикла вычисляется выражение, проверяется условие. Если условие истинно, вычисляется тело цикла.

Цикл выполняется до тех пор, пока указанное условие истинно.

Слайд 82

Существует много вариантов использования оператора цикла с параметром.

Рассмотрим примеры, демонстрирующие возможности использования этого оператора. Во всех примерах вычисляется сумма первых **n [эн]** натуральных чисел.

В первом примере переменная i [ай] описана внутри оператора, при инициализации. Данная переменная является локальной и доступна, видима, только в теле данного оператора. Тело цикла – простой оператор, фигурные скобки опущены.

Во втором примере есть элементы – инициализация и условие, а элемент – выражение отсутствует. Наличие символа – точки с запятой внутри круглых скобок обязательно. Этого требует синтаксис языка C++ [си плюс плюс]. Значение параметра цикла увеличивается на единицу в теле цикла, что фактически заменяет элемент – выражение.

Так как элемент – выражение выполняется после цикла, используется постфиксная форма увеличения значения переменной i [ай] на единицу.

Слайд 83

В первом примере на слайде продемонстрирован фрагмент программы вычисления суммы первых n [эн] натуральных чисел.

Для наглядности на слайде пронумерованы:

- один – инициализация:
- два – условие;
- три – выражение.

Данный оператор инициализирует начальные значения нескольких переменных, разделенных запятыми, это переменные i [ай] и s [эс].

Сумма чисел вычисляется в элементе – выражение, и фактически тело цикла в явном виде отсутствует. Наличие точки с запятой после круглой скобки обязательно. Фактически тело цикла – это пустой оператор.

Второй пример – это бесконечный цикл.

Приведенные примеры продемонстрировали, что любой элемент этого оператора, инициализация, условие или выражение, могут отсутствовать. Однако действия, предусмотренные этими элементами, все равно должны выполняться.

Слайд 84

В операторе цикла с параметром фактически задается закон изменения параметра цикла. При использовании оператора for [фо] необходимо точно знать, с каким значением параметра цикла завершается цикл.

На слайде приведен пример использования операторов цикла с параметром, чтобы продемонстрировать, с каким значением переменой i [ай] завершится первый оператор for [фо]. Значение переменной i [ай] при выходе из первого цикла равно трем. Согласно правилам организации работы этого оператора, значение параметра цикла успевает увеличиться на единицу, и только потом проверяется условие выполнения цикла.

На слайде демонстрируется результат выполнения этого фрагмента – первые три числа выведет на печать первый оператор for [фо], а следующие четыре числа – второй оператор. После вывода результатов работы первого оператора курсор остается на этой строке, и результаты работы второго оператора выводятся на этой же строке.

Обратите внимание на то, что у второго оператора цикла отсутствует элемент – инициализация, то есть используется значение переменной цикла i [ай] равное трем.

Рассмотренные примеры показали, что оператор цикла с параметром имеет большие возможности, использование которых облегчает оптимизирование процесса программирования.

Слайд 85

На слайде приведены два оператора.

Break [брэйк] – оператор прерывания цикла. Break [брэйк] означает «сломать». В соответствии с этим оператором прерывается выполнение цикла, и управление передается следующему оператору.

Оператор break [брэйк] приводит к завершению выполнения циклов do while [ду вайл], for [фо] или while [вайл].

Ранее мы уже рассматривали этот оператор, который использовался в структуре оператора выбора, или как его еще называют, операторе переключения – switch [свич].

Continue [конти́нюи] – этот оператор прерывает лишь очередную итера́цию цикла, как правило в результате проверки какого-либо условия, затем осуществляется переход к следующей итерации. То есть, цикл не прерывается.

Continue [конти́нюи] означает «продолжить». Это позволяет сразу перейти в конец тела цикла, пропуская весь код программы, который находится под ним. Это полезно в тех случаях, когда мы хотим завершить текущую итерацию раньше времени.

В случае с циклом for [фо] часть изменения параметра цикла по-прежнему выполняется, даже после выполнения оператора continue

[конти́нюи], так как изменение параметра цикла происходит вне тела цикла. Вспомните структуру этого оператора цикла.

«С формальной точки зрения операторы break [брэйк] и continue [конти́нюи] не являются операторами структурного программирования. Однако их использование в ограниченных количествах оправдано, когда они упрощают понимание программы и позволяют избегать больших вложенных структур. Например, мы проверяем входные данные на аномалии. Если не использовать эти операторы, то всю обработку придется вложить в условный блок, что ухудшает читабельность программы. Вместо этого можно написать небольшой условный блок, который организует выход из функции при неверных исходных данных» [3].

Слайд 86

На слайде представлен фрагмент программного кода, в котором используются вложенные циклы, когда в теле одного цикла, внешнего, размещается другой цикл, внутренний. Ограничений на вложенность циклов нет. Строить вложенные циклы можно с помощью любых операторов цикла, допустимых на языке С++ [си плюс плюс]. На слайде приведены результаты, которые будут выведены на экран во вложенных циклах.

Вложенные циклы строятся по тем же правилам, что и простые.

Нельзя менять значение параметра внешнего цикла в теле внутреннего. Нельзя войти в тело цикла с помощью оператора goto [гоу ту].

При построении вложенных циклов необходимо знать правила построения вложенных циклов. Параметр внешнего цикла меняется медленнее, чем параметры внутреннего цикла. При одном значении параметра внешнего цикла параметр внутреннего цикла пробегает все свои возможные значения.

На слайде введены обозначения:

– 1 [единица] – внутренний цикл;
– 2 [двойка] – внешний цикл.

Вложенные циклы не должны пересекаться, поэтому можно мысленно соединить конец и начало каждого цикла, как показано на слайде, и убедиться в том, что циклы не пересекаются. Иначе можно нарушить законы изменения параметров внешнего и внутреннего циклов.

Оператор break [брэйк] прерывает только тот цикл, в теле которого используется. В примере, приведенном на слайде, этот оператор прерывает внутренний цикл, передавая управление внешнему циклу, вернее, следующей итерации внешнего цикла.

Слайд 87

На слайде приведен фрагмент, демонстрирующий работу оператора contiuue [конти́нюи] на конкретном примере.

Данный фрагмент позволяет вывести на экран все четные числа в диапазоне от нуля до заданного n [эн], значение которого вводится с клавиатуры. Для определения чётности используется операция – остаток от деления. Эта операция применима только к целым переменным.

Каждый раз, когда значение параметра цикла является нечетным числом, прерывается очередная итерация и осуществляется переход к следующей итерации.

Конечно, это можно было реализовать непосредственно в операторе цикла, задав шаг изменения равным двум. Но для демонстрации работы оператора прерывания, использован способ, приведенный на слайде.

При использовании оператора continue [конти́нюи] с циклами do while [ду вайл] или while [вайл] надо быть осторожными. Поскольку в этих циклах изменение параметра цикла выполняется непосредственно в теле цикла,

использование этого оператора может привести к тому, что цикл станет бесконечным.

Еще раз напоминаем о том, что войти в цикл можно только сверху, нельзя войти в середину цикла, прервать цикл можно в любом месте.

Слайд 88

На слайде приведен фрагмент программы, реализующий следующую задачу – проездной билет на автобус состоит из шестизначного числа. Найти и вывести на печать номера счастливых билетов. Счастливым считается билет, у которого сумма первых трех цифр равна сумме вторых трех цифр.

Для решения задачи используем структуру вложенных циклов. Введены шесть переменных целого типа, которые являются параметрами циклов. Каждая из переменных определяет соответствующую цифру в номере билета. Необходимо имитировать номера билетов, то есть перебрать все возможные варианты и выбрать среди них номера счастливых билетов. Этот перебор позволяют выполнить вложенные циклы.

В каждой позиции шестизначного числа предусмотрены цифры в диапазоне от нуля до девяти. Номера билета, у которого все цифры нули, не существует. Поэтому с помощью оператора continue [конти́нюи] прерывается очередная итерация цикла, если сформировался билет, сумма цифр которого равна нулю.

Напомним, что указанный оператор прерывает только очередную итерацию цикла, передавая управление следующей итерации цикла, а именно циклу с параметром f [эф].

Конечно, не рекомендуется использовать большую вложенность циклов. Это может привести к ошибкам типа пересечения циклом, нарушения закона изменения параметров цикла и так далее.

Слайд 89

Тема 4. Указатели и адреса объектов

Одним из основных понятий языка C++ [си плюс плюс] является понятие объекта как некоторой области памяти. Тип переменной определяет совокупность значений, которые может принимать переменная, а также размер памяти в байтах, выделенной под хранение значений переменной.

При выполнении инициализации переменной ей автоматически присваивается свободный адрес памяти, и любое значение, которое мы присваиваем переменной, сохраняется по этому адресу памяти.

Прежде всего нужно разобраться с двумя терминами – область видимости и продолжительность. Область видимости определяет, где можно использовать переменную. Продолжительность, или как еще называют – «время жизни», определяет, где переменная создаётся и где уничтожается. Эти понятия тесно связаны между собой.

Переменные, определённые внутри блока, называются локальными переменными. Локальные переменные имеют автоматическую продолжительность и локальную область видимости. Они создаются и инициализируются в точке определения и уничтожаются при выходе из блока.

Для автоматических переменных память выделяется динамически – во время выполнения программы при входе в блок или функцию, и освобождается при выходе из блока или функции. Ключевое слово auto [ауто] присваивается по умолчанию.

Статические переменные хранятся в памяти. Они могут быть локальными и внешними. Статические переменные отличаются от автоматических переменных тем, что при выходе из блока или функции, в которых они были объявлены, значения статических переменных не теряются, выделенная память не освобождается. Они не доступны вне блока

или функции, в которыыыыыых объявлены. Это внутренний тип компоновки и статическая продолжительность существования.

Слайд 90

Внешние переменные – это внешний тип компоновки и статическая продолжительность существования.

Переменная, имеющая внешние связи, называется внешней переменной. Она может использоваться как в файле, в котором определена, так и в других файлах.

Если вы хотите сделать глобальную переменную внутренней, которую можно использовать только внутри одного файла, используйте ключевое слово static [стэ́тик]. Аналогично, если вы хотите сделать глобальную переменную внешней, которую можно использовать в любом файле, используйте ключевое слово extern [э́кстэрн].

Ключевое слово static [стэ́тик] можно использовать для объявления переменных и функций в глобальной области видимости, области пространства имен и области класса. Статические переменные также могут быть объявлены в локальной области видимости.

На слайде приведен пример использования статических переменных.

В данном примере используется оператор цикла с параметром. В структуре этого оператора описана и определена переменная, что означает, что этот оператор является блоком. Назовем его внешним блоком.

Первая переменная i [ай], описанная в структуре оператора for [фо], является локальной переменной, и сфера ее видимости – от начала и до конца работы этого оператора.

Вторая переменная i [ай] описана внутри еще одного блока, образованного фигурными скобками в теле цикла, назовем его внутренним

блоком. Эта переменная описана как статическая и сохраняет свою продолжительность до конца работы внешнего блока.

Приведенные результаты демонстрируют, что статическая переменная, описанная во внутреннем блоке, сохраняет значение до конца работы внешнего блока.

Если в данном фрагменте убрать во внутреннем блоке статическое описание переменной, то ее значение в блоке будет равным числу три при каждом прохождении через цикл.

Слайд 91

«Статическая длительность означает, что объект или переменная выделяется при запуске программы и освобождается при ее завершении. Внешняя компоновка означает, что имя переменной видно за пределами файла, в котором эта переменная объявлена. Внутренняя компоновка означает, что имя не видно за пределами файла, в котором объявлена переменная. По умолчанию объект или переменная, определенные в глобальном пространстве имен, имеют статическую длительность и внешнюю компоновку.

Ключевое слово static [ста́тик] можно использовать в следующих ситуациях:

- при объявлении static [ста́тик] переменной или функцией в области видимости файла. Ключевое слово указывает на то, что переменная или функция имеет внутреннюю компоновку. При объявлении переменной она имеет статическую длительность, и компилятор инициализирует ее со значением ноль, если не указано другое значение;
- при объявлении static [ста́тик] переменной в функции, ключевое слово указывает на то, что переменная удерживает свое состояние между вызовами этой функции;

– объявление членов объединения как статических невозможно. Однако глобально объявленное анонимное объединение должно быть явно объявлено static [стэ́тик]» [7].

В этом примере показано, как статическая переменная nStat [э́нстат], объявленная в функции, удерживает свое состояние между вызовами этой функции. На слайде указаны результаты вычислений с использованием статических переменных.

Локальные автоматически создаваемые объекты или переменные инициализируются каждый раз, когда поток элемента управления достигает их определения. Локальные статические объекты или переменные инициализируются, когда поток элемента управления достигает их определения в первый раз.

Слайд 92

Как говорилось ранее, все используемые в программе переменные должны быть описаны и определены. Согласно описанию определяется тип переменной, объем памяти, отводимой под хранение переменной.

На слайде показано, как описывать переменную указатель.

Указатель – это переменная, значением которой является адрес памяти, или ячейка памяти. Указатели объявляются точно так же, как и обычные переменные, только со звёздочкой между типом данных и идентификатором, Только инициализируется указатель не значением одного из множества типов данных языка С++ [си плюс плюс,], а адресом некоторой переменной, которая была объявлена ранее.

Нам не нужно беспокоиться о том, какие конкретно адреса памяти выделены для определенных переменных. Мы просто ссылаемся на переменную через присвоенный ей идентификатор, а компилятор

конвертирует это имя в соответствующий адрес памяти. Однако доступ к участкам памяти возможен и через указатели.

Значениями указателей служат адреса участков памяти, выделенных для объектов конкретных типов. В определении и описании указателя всегда присутствует обозначение соответственного ему типа. С помощью указателя можно получить доступ ко всему сохраняемому объекту в целом.

На слайде приведены примеры описания переменных. Две основные операции над переменными-указателями – это разыменование, символ *[звёздочка], и получение адреса, символ &[амперса́нд].

В качестве типа, который используется при объявлении указателя, можно выбрать тип void [войд]. Но в этом случае при инициализации указателя придется приводить его к типу переменной, на которую он указывает.

Не следует путать оператор взятия адреса со ссылкой на некоторое значение, которое так же визуально отображается символом &[амперса́нд] .

Указатели делятся на указатели на объекты и на указатели на функции.

С указателями на функции мы познакомимся позднее.

Слайд 93

На слайде приведен фрагмент обращения к одним и тем же ячейкам памяти через указатель и через имя переменной, через идентификатор.

Синтаксически язык C++ [си плюс плюс] принимает объявление указателя, когда звёздочка находится рядом с типом данных, с идентификатором или даже посередине. Обратите внимание, эта звёздочка не является оператором разыменования. Это всего лишь часть синтаксиса объявления указателя.

Указатели должны иметь тип данных. Без типа указатель не знал бы, как интерпретировать содержимое, на которое он указывает при

разыменовании. Поэтому и должны совпадать тип указателя с типом переменной. Если они не совпадают, то указатель при разыменовании может неправильно интерпретировать биты. Например, вместо вещественного типа double [дабл] использовать целочисленный тип int [инт]. Под хранение переменных целого и вещественного типа отводится одинаковое количество байт – по четыре байта.

Однако при объявлении нескольких указателей звёздочка должна находиться возле каждого идентификатора. Это легко забыть, если вы привыкли указывать звёздочку возле типа данных, а не возле имени переменной.

При объявлении указателя рекомендуется указывать звёздочку возле имени переменной. Как и обычные переменные, указатели не инициализируются при объявлении. Содержимым неинициализированного указателя является обычный мусор.

Оператор разыменования выглядит так же, как и операция умножения. Отличить их можно по тому, что операция разыменования – уна́рная операция, а операция умножения – бинарная. Как известно, в уна́рных операциях участвует один операнд, а в бина́рных операциях участвуют два операнда.

Слайд 94

«Так как указатель есть адрес, а адреса распределяет сам компьютер, то указатель не может быть инициализирован непосредственно. Знак &[амперса́нд] перед именем переменной возвращает ее адрес, или, как говорят, дает ссылку. Для того чтобы получить значение переменной, производится операция разыменования. Знак *[звёздочка] перед указателем возвращает значение той переменной, на которую указатель ссылается.

Указатели в языке C++ [си плюс плюс] по своей природе являются небезопасными, а их неправильное использование – один из способов получить сбой программы.

При разыменовании указателя программа пытается перейти в ячейку памяти, которая хранится в указателе, и извлечь содержимое этой ячейки. По соображениям безопасности современные операционные системы запускают программы для предотвращения их неправильного взаимодействия с другими программами. А также для защиты стабильности самой операционной системы» [7].

Рассмотрим на слайде фрагмент некорректного использования указателя. Если программа попытается получить доступ к ячейке памяти, не выделенной для нее операционной системой, то сразу завершится выполнение этой программы. Переменная – указатель p [пе] не описана и не определена, не известен адрес памяти, по которому происходит обращение.

Фрагмент программы на слайде иллюстрирует сказанное. При запуске вы получите сбой.

Слайд 95

Размер указателя зависит от архитектуры, на которой скомпилирован исполняемый файл – тридцати двухбитный исполняемый файл использует тридцати двухбитные адреса памяти. Следовательно, указатель на 32-битном [тридцати двухбитном] устройстве занимает 32 бита, что составляет 4 байта. С 64-битным [шестидесяти четырехбитным] исполняемым файлом указатель будет занимать 64 бита, что составляет 8 байт. И это вне зависимости от типа информации, на которую указывает переменная –указатель.

На слайде приведены примеры указателей на объекты разного типа: символьный тип, целочисленный тип и вещественный тип. Размер памяти,

отводимой под хранение переменных перечисленных типов, соответственно, различен.

Используя функцию sizeof [сайз оф] для определения размера указателей на переменные разного типа, можно увидеть, что размер указателя всегда один и тот же. Это связано с тем, что указатель – это всего лишь адрес памяти, а количество бит, необходимое для доступа к адресу памяти на определенном устройстве, всегда постоянное.

Предлагаем с помощью функции sizeof [сайз оф] определить количество байт, отводимых под указатели на вашем устройстве. Примените проверку на указателях разного типа, чтобы убедиться в изложенных положениях.

«Несмотря на то, что внутри функции значения переменных могут поменяться, по окончании работы в основной программе они остаются неизменными. С данной проблемой можно справиться, используя передачу аргументов по ссылке с помощью указателей. При передаче указателя на самом деле передается лишь адрес объекта, а, следовательно, функция получает возможность манипулировать значением, находящимся по этому адресу. Этот факт также свидетельствует о возможностях использования указателей. Подробнее о функциях на языке C++ [си плюс плюс] мы поговорим в последующих темах» [7].

Слайд 96

«Динамические переменные создаются с помощью специальных функций и операций. Они существуют либо до конца работы программы, либо до тех пор, пока не будет освобождена выделенная под них память с помощью специальных функций или операций. То есть время жизни динамических переменных – от точки создания до конца программы или до явного освобождения памяти» [3].

Над указателями допустимы операции, приведенные в таблице.

При выполнении инициализации переменной, ей автоматически присваивается адрес памяти, и любое значение, которое мы присваиваем переменной, сохраняется по этому адресу памяти.

Операции разыменования позволяют узнать, какой адрес памяти присвоен определённой переменной. Поскольку указатели содержат только адреса, то при присваивании указателю значения – это значение должно быть адресом.

Как только у нас есть указатель, указывающий на что-либо, мы можем применить к нему операцию разыменования, чтобы получить значение, на которое он указывает. После операции разыменования указатель позволяет обратиться к содержимому ячейки памяти, в которой находится конкретная информация.

Слайд 97

На слайде приведены примеры описания и определения указателей на переменные целого типа и переменные вещественного типа. Доступ к информации возможен как через имена переменных, так и через указатели.

Указатель должен иметь такой же тип, какой имеет объект, на который он указывает. Вычитая два объекта одного типа, можно определить расстояние между двумя участками памяти. Расстояние определяется в единицах, кратных длине в байтах объекта того типа, к которому отнесен указатель.

Разность однотипных указателей, содержащих адреса двух смежных объектов любого типа по абсолютной величине всегда равна единице.

На слайде продемонстрированы примеры с указателями на переменные разного типа, переменные целого типа и переменные вещественного типа. Разница указателей, в обоих случаях, равна единице.

При использовании переменной – указателя подразумевается работа с адресами, если перед именем переменной – указателя ставится звездочка – происходит обращение к информации, записанной по данному адресу.

Слайд 98

Язык C++ [си плюс плюс] поддерживает три основных типа выделения, распределения памяти, с двумя из которых мы уже знакомы.

Статическое выделение памяти выполняется для статических и глобальных переменных. Память выделяется один раз при запуске программы и сохраняется на протяжении работы всей программы.

«Автоматическое выделение памяти выполняется для параметров функции и локальных переменных. Память выделяется при входе в блок, в котором находятся эти переменные, и удаляется при выходе из него.

Динамическое выделение памяти является выделением памяти из операционной системы по требованию. При использовании динамической памяти появляются проблемы, связанные с тем, что любое выделение или освобождение памяти – это системный вызов, замедляющий работу программы.

Как для статического, так и для автоматического распределения памяти размер переменной должен быть известен во время компиляции. Память для обычных переменных выделяется из специального резервуара памяти – стека. Объём памяти стека в программе, как правило, невелик.

Эти проблемы легко устраняются с помощью динамического выделения памяти. Эта память выделяется из большего хранилища, управляемого операционной системой, – «ку́чи». На современных компьютерах размер «кучи» может составлять гигабайты памяти» [4].

«Правильное понимание и использование указателей имеет большое значение при создании большинства программ по четырем причинам:

– указатели предоставляют способ, позволяющий функциям модифицировать передаваемые аргументы;
– указатели используются для поддержки системы динамического выделения памяти;
– использование указателей может повысить эффективность работы некоторых подпрограмм;
– указатели, как правило, используются для поддержки некоторых структур данных типа связанные списки и двоичные деревья» [3].

Слайд 99

«Рассмотрим способы выделения памяти, динамического выделения памяти, связи указателей и динамического распределения памяти.

Существует два основных способа хранения информации в оперативной памяти. Первый заключается в использовании глобальных и локальных переменных. В случае глобальных переменных выделяемые под них поля памяти остаются неизменными на все время выполнения программы. Под локальные переменные программа отводит память из сте́кового пространства. Однако локальные переменные требуют предварительного определения объема памяти, выделяемой для каждой ситуации. Хотя C++ [си плюс плюс] эффективно реализует такие переменные, от программиста требуется заранее знать, какое количество памяти необходимо для каждой ситуации.

Второй способ, которым C++ [си плюс плюс] может хранить информацию, заключается в использовании системы динамического распределения. При этом способе память распределяется для информации из свободной области памяти по мере необходимости. Область свободной памяти находится между кодом программы с ее постоянной областью памяти

и стеком. Динамическое размещение удобно, когда неизвестно, сколько элементов данных будет обрабатываться.

По мере использования программой стековая область увеличивается вниз, то есть программа сама определяет объем стековой памяти. Например, программа с большим числом рекурсивных функций займет больше стековой памяти, чем программа, не имеющая рекурсивных функций, так как локальные переменные и возвращаемые адреса хранятся в сте́ках. Память под саму программу и глобальные переменные выделяется на все время выполнения программы и является постоянной для конкретной среды» [5].

Память, выделяемая в процессе выполнения программы, называется динамической. После выделения динамическая память сохраняется до ее явного освобождения, что может быть выполнено только с помощью специальной операции или библиотечной функции.

Слайд 100

Если динамическая память не освобождена до окончания программы, то она освобождается автоматически при завершении программы. Тем не менее явное освобождение ставшей ненужной памяти является признаком хорошего стиля программирования.

В процессе выполнения программы участок динамической памяти доступен везде, где доступен указатель, адресующий этот участок. Таким образом, возможны следующие три варианта работы с динамической памятью, как представлено на слайде.

«Все переменные, объявленные в программе, размещаются в одной непрерывной области памяти, которую называют сегментом данных. Такие переменные не меняют своего размера в ходе выполнения программы и называются статическими. Размера сегмента данных может быть недостаточным для размещения больших объемов информации. Выходом из

этой ситуации является использование динамической памяти. Динамическая память – это память, выделяемая программе для ее работы за вычетом сегмента данных, стека, в котором размещаются локальные переменные подпрограмм и собственно тела программы.

Для работы с динамической памятью используют указатели. С их помощью осуществляется доступ к участкам динамической памяти, которые называются динамическими переменными. Для хранения динамических переменных выделяется специальная область памяти, называемая «кучей».

Динамические переменные создаются с помощью специальных функций и операций. Они существуют либо до конца работы программы, либо до тех пор, пока не будет освобождена выделенная под них память с помощью специальных функций или операций. То есть время жизни динамических переменных – от точки создания до конца программы или до явного освобождения памяти» [5].

Слайд 101

Для динамического выделения памяти для одной переменной используется оператор new [нью].

В примере выделяется память для целочисленной переменной. Оператор new [нью] возвращает указатель, содержащий адрес выделенной памяти. Для доступа к выделенной памяти создаётся указатель. Затем мы можем разыменовать указатель для получения значения.

В выделенный участок заносится значение, определяемое инициализатором, который не является обязательным элементом. В случае успешного выполнения оператор new [нью] возвращает адрес начала выделенного участка памяти. Если участок нужных размеров не может быть выделен, то оператор new [нью] возвращает нулевое значение адреса.

Когда с этой памятью уже выполнено все, что было необходимо, то её можно вернуть обратно в операционную систему. В отличие от статического или автоматического выделения памяти, программа самостоятельно отвечает за запрос и обратный возврат динамически выделенной памяти. Для динамических переменных это выполняется с помощью оператора delete [дели́т].

Этот оператор на самом деле ничего не удаляет. Он просто возвращает память, которая была выделена ранее, обратно в операционную систему. Затем операционная система может переназначить эту память для любой другой переменной. Оператор new [нью] чаще всего используется для размещения в памяти данных определенных пользователем типов, например, структур.

При выделении динамической памяти для массива его размеры должны быть полностью определены. Такая операция позволяет выделить в динамической памяти участок для размещения массива соответствующего типа, но не позволяет его инициализировать. В результате выполнения данного фрагмента оператор new [нью] возвратит указатель, значением которого служит адрес первого элемента массива.

Слайд 102

«Операции new [нью] и delete [дели́т] служат для выделения и освобождения блоков памяти. Область памяти, в которой размещаются эти блоки, называется свободной памятью. Операция new [нью] позволяет выделить и сделать доступным свободный участок в основной памяти, размеры которого соответствуют типу данных, определяемому именем типа.

В случае успешного выполнения операция new [нью] возвращает адрес начала выделенного участка памяти.

Если участок нужных размеров не может быть выделен, нет памяти, то операция new [нью] возвращает нулевое значение адреса» [3].

Рассмотрим, в каких случаях полезно использование указателей на различные объекты.

Указатели являются единственным способом динамического выделения памяти на языке C++ [си плюс плюс]. Это, безусловно, самый распространённый вариант использования указателей. Указатели можно эффективно использовать при обработке массивов. Указатели могут использоваться для итерации по массиву. Они могут заранее не определять размер памяти, выделенной под хранение элементов массива, выделять и освобождать память под массив непосредственно во время выполнения программы.

Подведем итоги. Перечислим все преимущества использования переменных – указателей.

Указатели могут использоваться для передачи большого количества данных в функцию без копирования этих данных, могут использоваться для передачи одной функции в качестве параметра другой функции.

Понятие указателя будет проходить тонкой линией во всех последующих темах, которые мы будем изучать – и в массивах, и в функциях, и в символьных массивам, и в файлах, и наконец, полностью раскроются при изучении динамических структур данных. Указатели могут использоваться для представления одной структуры в другую структуру, формируя, таким образом, целые цепочки, динамические структуры данных.

О динамических массивах, вернее о выделении динамической памяти под хранение элементов массива, мы поговорим далее.

Слайд 103

Тема 5.1. Массивы (Часть 1)

Обработка большого объема информации приводит к необходимости использования массивов.

Массив – это поименованный набор однотипной информации.

Массив – это группа элементов одинакового типа. В дальнейшем мы рассмотрим возможности хранения информации разного типа в одном массиве с помощью пользовательских типов данных, структур.

«Напомним, что массив – это упорядоченный набор величин, обозначаемых одним именем. Данные, являющиеся элементами массива, располагаются в памяти компьютера в определенном порядке, который задается индексами, порядковыми номерами элементов массива. В С++ [си плюс плюс] массив, как и любая переменная, должен быть объявлен. Делается это с помощью служебного слова, указывающего тип. Смешанные массивы не допустимы» [7].

Имя массива – уникальный идентификатор, строится по тем же правилам, что и имена простых переменных. Тип элементов массива определяет количество байт, необходимое для хранения каждого элемента.

В соответствии с объявлением массива выделяется память под его хранение. Имя массива – это фактически, указатель, который хранит адрес начала области памяти, выделенной под массив.

Весь набор информации объединен общим именем, а доступ к отдельным элементам массива осуществляется с помощью индекса. Индекс указывается рядом с именем массива в квадратных скобках. Индексом может быть числовая константа, переменная или выражение целого типа.

«При работе с массивами необходимо внимательно следить за тем, чтобы не выходить за их объявленные границы. Компилятор С++ [си плюс плюс], в отличие, например, от Паскаля, не предупреждает об этой ошибке.

Попытка ввести больше элементов, чем описано, приведет к неверным результатам, а попытка вывести – выведет случайный результат, находящийся в памяти» [7].

Слайд 104

На слайде описаны и инициализированы три массива. Определены начальные значения элементов массива. Индексы массивов начинаются с нуля, то есть в массиве всегда присутствует нулевой элемент.

Массив a [а] – это массив, целых чисел, состоящий из пяти элементов. Значения первых четырех элементов этого массива определены, элементы четвертый и пятый – получают нулевые значения.

Массив b [би] – это символьный массив, его размер в явном виде не задан, о чем свидетельствуют пустые квадратные скобки рядом с именем массива. Размер будет определен количеством элементов при начальной инициализации, то есть массив b [би] состоит из четырех элементов.

Массив c [си] – также массив символов. При определении элементов массива, как показано в третьем примере, в конце цепочки символов автоматически ставится признак конца строки символов – слеш ноль. То есть в массиве c [си] – пять элементов.

Если размер массива при описании отсутствует, список начальных значений массива в определении массива обязателен.

С помощью операции sizeof [сайз оф] можно определить размер массива. В формуле:

- делимое – имя массива, определяет размер всего массива в байтах;
- делитель – элемент массива, например, первый, определяет размер в байтах одного элемента массива.

По этой формуле определяется размер массива, то есть количество элементов в массиве.

Слайд 105

Как уже говорилось, при обращении к элементам массива используют индексы. Индекс определяет не номер элемента, а его смещение относительно начала массива. Имя массива с индексом в квадратных скобках фактически является выражением с двумя операндами:

- имя массива – это первый операнд, адрес начала массива в памяти;
- индекс – второй операнд, выражение целого типа, определяющее смещение от начала массива. Смещение на определенное количество байт относительно начала массива определяется типом элементов массива.

Напомним, что операция *«звездочка» позволяет получить значение объекта по его адресу. На слайде продемонстрированы две записи абсолютно идентичные. Приведены примеры, демонстрирующие два варианта обращения к массиву – через индекс и через указатель.

Во второй строке таблицы продемонстрировано обращение к элементам массива, используются индексы. Цикл продолжается, пока не встретится символ – признак конца строки.

В третьей строке таблицы обращение к элементам массива происходит через указатель. Имя массива и есть указатель на начало массива. А индекс – смещение относительно начала массива.

Слайд 106

На слайде приведены два примера вывода на экран значений элементов массива.

В первом примере описан массив и определены его значения. Вывод элементов осуществляется в цикле, в котором перебор элементов осуществляется с помощью индекса массива.

Во втором примере использован оператор цикла foreach [фóрэйч].

Новый тип цикла – foreach [фóрэйч] – цикл, основанный на диапазоне, предоставляет более простой и безопасный способ итерации по массиву.

При выполнении цикла переменной number [нáмбэ] присваивается значение первого элемента, т. е. значение ноль. Дальше программа выполняет вывод значения переменной number [нáмбэ] Затем цикл выполняется снова, и значением number [нáмбэ] уже является второй элемент массива. Вывод значения number [нáмбэ] выполняется снова. Цикл продолжает своё выполнение до тех пор, пока в массиве не останется не пройденных элементов.

Обратите внимание на то, что переменная number [нáмбэ] не является индексом массива. Ей просто присваивается значение элемента массива в текущей итерации цикла.

Слайд 107

Рассмотрим примеры обработки массивов. На слайде – формулировка задачи.

В условии задачи указано, что имеется массив целых чисел.

Для нахождения среднего значения элементов массива введена новая переменная positive [пóзитив], которая должна иметь вещественный тип. Данная переменная описана как вещественная, чтобы при подсчете среднего значения элементов массива, то есть при делении на количество элементов не исказился результат. Известно, что при делении целочисленных переменных, дробная часть теряется.

Переменные, описанные в программе, автоматически не обнуляются. Поэтому необходимо предварительно обнулить значение переменной positive [пóзитив].

В данной задаче реализуется ввод размера массива и значений элементов массива с клавиатуры.

Ввод осуществляется в цикле, причем, каждое прохождение через цикл соответствует вводу одного элемента массива.

Если в массиве отсутствуют положительные числа, вывести на печать соответствующее сообщение.

Слайд 108

На слайде первый фрагмент программы. С клавиатуры вводится конкретный размер массива n [эн]. Описан массив, элементами которого являются целые числа. Размер массива заранее не определяется, чтобы избежать избыточного выделения памяти под хранение элементов массива.

Идентификатор array [э́ррэй] – имя массива.

В отличии от некоторых алгоритмических языков, на языке С++ [си плюс плюс] значения переменных, описанных в программе, автоматически не обнуляются. Поэтому, переменную positive [позитив], в которой будет подсчитываться среднее значение положительных элементов массива, необходимо обнулить.

Об этом факте не следует забывать, чтобы не произошло искажения результатов вычислений.

Слайд 109

На слайде демонстрируется продолжение программы.

Обратите внимание на то, что в одном цикле предусмотрен ввод элементов массива и подсчет суммы положительных элементов массива. Используется оператор цикла с параметром. Последовательность операторов, образующих тело цикла, необходимо заключить в фигурные скобки.

Параметр цикла – переменная i [ай], описана в структуре оператора цикла, то есть тело цикла является блоком. Об этом мы уже говорили ранее.

Не следует забывать, что переменная, описанная внутри блока, не видна вне этого блока, и время ее существования – от начала и до конца работы данного блока.

После выполнения цикла анализируется значение переменной positive [по́зитив], и если оно равно нулю, выдается сообщение об отсутствии в массиве положительных чисел. Данное действие реализует свойство алгоритма – результативность, то есть или программа печатает результат, или сообщение о причинах отсутствия результата.

Слайд 110

Возможность генерировать случайные числа очень полезна в некоторых задачах, в частности в программах научного или статистического моделирования.

Компьютеры не способны генерировать случайные числа. Вместо этого они могут имитировать случайность, что достигается с помощью генераторов псевдослуча́йных чисел.

Функция rand [ранд] генерирует случайное число в последовательности.

В большинстве случаев нам не нужны рандо́мные числа между 0 [нулем] и максимально возможными числами. Нам нужны числа между двумя другими значениями. Например, если нам нужно сымитировать бросок кубика, то диапазон значений будет невелик: от 1 до 6 [от единицы до шести].

Чтобы сгенерировать числа в диапазоне от A до B [от эй до би] включительно, можно использовать формулу, указанную в таблице.

Слайд 111

На слайде продемонстрирован фрагмент программы, использующий генератор случайных чисел.

Задавая предельные значения, минимальное и максимальное, можно сгенерировать определенное количество случайных чисел из указанного диапазона.

Генератор случайных чисел – это программа, которая принимает стартовое, начальное значение и выполняет с ним определённые математические операции, чтобы конвертировать его в другое число, совсем не связанное со стартовым. Затем программа использует новое сгенерированное значение и выполняет с ним те же математические операции, что и с начальным числом, чтобы конвертировать его в ещё в одно новое число – третье. Применяя этот алгоритм к последнему сгенерированному значению, программа может генерировать целый ряд новых чисел, которые будут казаться случайными.

Если вы запустите программу несколько раз, то заметите, что в результатах всегда находятся одни и те же числа. Это означает, что, хотя каждое число в последовательности кажется случайным относительно предыдущего, вся последовательность не является случайной вообще. А это, в свою очередь, означает, что наша программа полностью предсказуема. Одни и те же значения ввода приводят к одним и тем же значениям вывода. Бывают случаи, когда это может быть полезно или даже желательно.

Слайд 112

Рассмотрим задачу поиска максимального элемента в одномерном массиве. С дополнительным условием – среди отрицательных элементов.

При решении подобных задач необходимо обратить внимание на то, какое начальное значение необходимо присвоить переменной, которой

присваивается значение максимального элемента. Если бы не было дополнительного условия – поиск максимального среди отрицательных элементов массива, то достаточно этой переменной присвоить первоначально значение любого из элементов массива, например, первого элемента.

В нашей задаче этой переменной необходимо присвоить значение элемента массива из множества отрицательных элементов массива. Это важно.

На слайде приведена таблица введенных переменных. Возможен вариант, когда в массиве отрицательные элементы отсутствуют. Именно по этой причине введена переменная flag [флаг], которая является признаком наличия или отсутствия в массиве отрицательных элементов.

Слайд 113

Переменная, отвечающая за максимальный элемент, должна иметь тот же тип, что и элементы массива.

Переменной flag [флаг], которая является признаком присутствия в массиве отрицательных чисел, первоначально присвоено значение равное нулю.

После перебора элементов массива, если значение переменной flag [флаг] осталось равным нулю, приходим к выводу о том, что в массиве отрицательные элементы отсутствуют.

В данной программе описан массив на сто элементов. С клавиатуры вводится количество элементов массива, и это число не может превышать числа «сто». Ввод конкретного количества элементов n [эн] позволяет отладить программу на меньшем количестве элементов.

Если появляется необходимость обработки большего количества элементов, то есть массива большего размера, необходимо внести изменения в программе.

Слайд 114

На слайде – фрагмент ввода элементов массива с клавиатуры. Ввод реализован с помощью оператора цикла с параметром.

В теле цикла присутствуют два оператора – ввод элементов и анализ на наличие отрицательных элементов. При вводе отрицательных чисел в переменную, отвечающую за максимальное значение, всему массиву присваивается значение отрицательного элемента. Не важно, какой это элемент, – первый или последний. Главное, что он из множества отрицательных чисел.

При этом переменной flag [флаг] присваивается значение равное единице, свидетельствующий о наличии в массиве отрицательных элементов.

Обратите внимание на то, что номер элемента также сохраняется.

Вспомним о свойстве алгоритма – результативности. То есть, алгоритм должен приводить или к результату, или к сообщению о причинах его отсутствия. Для реализации этого свойства алгоритма и введена переменная flag [флаг].

Слайд 115

На слайде – фрагмент поиска максимального среди отрицательных элементов массива.

Анализируется значение переменной flag [флаг], и, если в массиве есть отрицательные числа, начинается поиск максимального элемента среди отрицательных чисел. Сохраняется номер этого элемента, то есть определяется местоположение максимального отрицательного элемента в общем наборе.

В противном случае выдается сообщение об отсутствии отрицательных чисел в массиве.

Вывод результата предусматривает вывод номера и значения максимального элемента. Элементы одной строки выводятся на одной стороне экрана. Для того чтобы числа на слипались, используется управляющий символ \t [обратный слеш ти].

Обратите внимание на то, что для большей наглядности при выводе на печать рядом с именем массива в квадратных скобках выводится индекс найденного элемента.

Слайд 116

Указатель – это переменная, значением которой является адрес памяти, ячейка памяти.

Указатели объявляются точно так же, как и обычные переменные, только со звездочкой между типом данных и идентификатором.

Поскольку указатели содержат только адреса, то при присваивании указателю значения следует помнить, что это значение должно быть адресом. Указатели должны иметь тип данных. Без типа указатель не знал бы, как интерпретировать содержимое, на которое он указывает при разыменовании. Об этом мы уже говорили ранее.

Память для динамических массивов выделяется из иного резервуара, чем память, используемая для фиксированных массивов. Размер массива может быть довольно большим.

Память для большинства обычных переменных, включая фиксированные массивы, выделяется из специального резервуара памяти – сте́ка. Объём памяти стека в программе невелик. Если произойдёт переполнение стека, операционная система автоматически завершит выполнение вашей программы.

Эти проблемы легко устраняются с помощью динамического выделения памяти. Динамическое выделение памяти – это способ запроса

памяти из операционной системы запущенными программами при необходимости. Эта память выделяется не из ограниченной памяти стека программы, а из гораздо большего хранилища, управляемого операционной системой, – «ку́чи». На современных компьютерах размер «кучи» может составлять гигабайты памяти.

Использование динамических массивов очень удобно, когда появляется передача массива в функцию, об этом мы поговорим позднее.

На слайде показано объявление и удаление динамических массивов с помощью операторов new [нью] и delete [дели́т].

Слайд 117

«Существует два основных способа хранения информации в оперативной памяти. Первый заключается в использовании глобальных и локальных переменных. В случае глобальных переменных, выделяемая под них память, остается неизменной на все время выполнения программы.

Под локальные переменные программа отводит память из стекового пространства. Однако локальные переменные требуют предварительного определения объема памяти, выделяемой для каждой ситуации. Хотя язык C++ [си плюс плюс] эффективно реализует такие переменные, они требуют от программиста заранее знать, какое количество памяти необходимо для каждой ситуации.

Второй способ, которым C++ может хранить информацию, заключается в использовании системы динамического распределения. При этом способе память распределяется для информации из свободной области памяти по мере необходимости. Область свободной памяти находится между кодом программы с ее постоянной областью памяти и стеком. Динамическое размещение удобно, когда неизвестно, сколько элементов данных будет обрабатываться» [3].

Если до начала разработки программы неизвестно, сколько в массиве элементов, в программе следует использовать динамические массивы. Память под них выделяется с помощью операции new [нью] в динамической области памяти. Адрес начала массива хранится в переменной, называемой указателем имени массива.

В задаче будем использовать динамический массив. Динамическое выделение памяти для массива на языке C++ [си плюс плюс] позволяет устанавливать его длину во время выполнения программы.

Непосредственно во время выполнения программы можно и освобождать память, отведенную под хранение этого массива. В предыдущем слайде были указаны эти операторы.

Алгоритм предусматривает сравнение соседних элементов массива, и, если предыдущий элемент окажется больше последующего, то эти элементы необходимо поменять местами. Для этого будем использовать функцию swap [своп].

В последней строке таблицы приведена структура функции swap [своп], которая меняет местами содержимое двух переменных, двух аргументов этой функции. Аргументы указываются рядом с именем функции в скобках и разделяются запятой.

Слайд 118

«Очевидно, что элементы массива, как правило, располагаются в произвольном порядке. Но во многих случаях может понадобиться расположить их, например, в порядке возрастания. Такая процедура упорядочения называется сортировкой» [7].

Существуют различные алгоритмы сортировки. Но эти алгоритмы мы будем подробно изучать в рамках другой дисциплины.

Рассмотрим задачу, в которой требуется отсортировать элементы одномерного массива в порядке возрастания.

Необходимо преобразовать массив таким образом, чтобы каждый предыдущий элемент массива, был меньше последующего элемента.

Переменная n [эн]определяет размер массива.

Используем динамический массив. Размер массива заранее не определен. С помощью оператора new [нью] описан динамический массив.

Согласно этому описанию, выделяется память под хранение элементов массива. Заметим, что обнуление памяти при ее выделении не происходит. Инициализировать динамический массив нельзя. Обратиться же к элементу динамического массива можно. Как обычно с помощью индекса, так и с использованием указателя. Дело в том, что в переменном указателе хранится адрес начала массива. Размер выделенной памяти под размещение массива, зависит от типа элементов массива и размера массива, то есть от количества элементов.

Каждый раз при запуске программы на выполнение вводится размер массива и элементы массива с клавиатуры.

Для реализации процесса сортировки введена дополнительная переменная – flag [флаг]. Тип переменной определен как логический. Эта переменная будет использована как признак упорядоченности элементов массива. Напомним, логические переменные могут принимать одно из двух значений – «истина» или «ложь». На основе этих свойств логических переменных и построен алгоритм решения поставленной задачи.

Слайд 119

Для сортировки элементов массива используется структура вложенных циклов.

Введены обозначения:

– первое – внутренний цикл;

– второе – внешний цикл.

Внутренний цикл, реализованный оператором цикла с параметром, позволяет один раз пройтись по массиву, сравнивая соседние элементы.

Обратите внимание на конечное значение параметра внутреннего цикла в операторе for [фо]. На слайде это выражение подчеркнуто. Последний раз в цикле сравнивается предпоследний и последний элементы массива.

Перед каждым прохождением по массиву, переменной flag [флаг] присваивается значение «истина», предполагая, что массив упорядочен. Если предыдущий элемент окажется больше последующего, то эти элементы меняются местами, и выставляется признак того, что массив еще не отсортирован. Переменной flag [флаг] присваивается значение «ложь».

Внешний цикл, который описан с помощью оператора цикла с постусловием, продолжается до тех пор, пока все элементы не будут упорядочены по возрастанию.

Если необходимо упорядочить элементы по убыванию, достаточно в этом фрагменте в структуре оператора проверить условия, поменять знак «больше» на знак «меньше».

Если массив отработан, можно освободить память, отведенную под массив с помощью оператора delete [дели́т].

Для обмена элементов местами будем использовать функция swap [своп]. У этой функции два аргумента – соседние элементы массива, значения которых меняются местами.

Слайд 120

Тема 5.2. Массивы (Часть 2)

При решении определенных задач, одним из главных моментов является вопрос о том, как структурировать информацию, как правильно ее организовать.

Двумерный массив можно рассматривать как матрицу с определенным количеством строк и столбцов.

Весь набор информации объединяется общим именем. Доступ к отдельным элементам осуществляется с помощью двух индексов, номера строки и номера столбца, на пересечении которых находится этот элемент. Причем первый индекс, используемый при обращении к элементам массива, – всегда номер строки, а второй – всегда номер столбца. Индексами двумерного массива могут быть переменные, выражения или константы целого типа.

Двумерные массивы должны быть описаны в программе, указан тип элементов и количество элементов, что определяет объем выделяемой памяти под хранение элементов массива.

Надо помнить о том, что в двумерном массиве всегда есть нулевая строка и нулевой столбец.

Слайд 121

В двумерных массивах есть понятие главной и побочной диагоналей. На слайде эти диагонали обозначены.

Для решения многих задач бывает полезно знание некоторых соотношений между индексами массивов.

Двумерный массив можно рассматривать как матрицу. Если количество строк равно количеству столбцов, матрица называется квадратной.

В таблице приведены формулы, которые определяют соотношения между индексами элементов, которые расположены на главной диагонали. Соответственно, определены соотношения индексов элементов, расположенных над главной диагональю и под главной диагональю. Знание этих соотношений между индексами двумерного массива, помогает в решении многих задач.

При обозначении индексов элементов двумерного массива на первом месте всегда указывается номер строки, на втором месте всегда указывается номер столбца.

Слайд 122

На слайде приведены примеры описания и определения элементов двух массивов. Обратите внимание на то, что последовательность значений, которые присваиваются элементам массива, регулируется фигурными скобками и запятыми между фигурными скобками.

Известно, что на языке С ++ [си плюс плюс] переменные, описанные в программе, автоматически не обнуляются. Но при этом надо помнить и о том, что достаточно определить значения только некоторых элементов массива. Тогда остальные элементы массива будут автоматически обнулены.

В первом примере описан двумерный массив целых чисел. Определены значения элементов первого столбца. Распределение значений элементов обеспечивается с помощью фигурных скобок.

Во втором примере описан массив вещественных чисел. Обратите внимание на то, что количество строк в явном виде не указано. Об этом свидетельствуют пустые квадратные скобки рядом с именем массива. Это вполне допустимо. Количество строк будет определено присвоенными значениями элементам массива. Количество строк во втором массиве равно двум.

В нижней строке таблицы приведены значения, присвоенные элементам массива.

Слайд 123

На слайде представлены два варианта ввода элементов двумерного массива – по строкам и по столбцам.

Не столь важно, как вводятся элементы массива, по строкам или по столбцам. Необходимо знать, что элементы массива хранятся в памяти по строкам. То есть, сначала элементы первой строки, затем второй строки и так далее.

Ввод элементов двумерного массива выполняется во вложенных циклах. Способ ввода элементов массива по строкам или по столбцам обусловлен законом изменения индексов строк и индексов столбцов во внутреннем и внешнем циклах.

Известно, что во вложенных циклах индекс внешнего цикла меняется медленнее, чем индекс внутреннего цикла. При одном значении параметра внешнего цикла, параметр внутреннего пробегает все свои возможные значения. Именно это правило и позволяет организовать ввод элементов двумерного массива по строкам или по столбцам.

В двумерных массивах всегда присутствуют нулевая строка и нулевой столбец. На следующих слайдах приведены примеры обработки двумерных массивов.

Слайд 124

На слайде сформулирована задача.

Симметричной считается матрица, у которой элементы по разные стороны главной диагонали равны. То есть, если мысленно сложить матрицу по линии главной диагонали, элементы должны совпасть.

На слайде приведена квадратная матрица. Проанализировав индексы элементов по разные стороны главной диагонали, определим, какие элементы подлежат сравнению. В сравнении участвуют элементы, у которых индексы зеркально меняются.

Пример симметричной матрицы с конкретными числами приведен на слайде.

Введем переменную логического типа, которая будет принимать значение «истина», если матрица симметрична, и принимать значение «ложь», если матрица не симметрична.

Перебор элементов матрицы будет производиться во вложенных циклах. Для перебора элементов двумерного массива удобно использовать оператор цикла с параметром, который автоматически будет менять индексы массива.

Слайд 125

На слайде приведен фрагмент программы, позволяющий определить симметричность квадратной матрицы. Обратите внимание на то, что и внутренний цикл, и внешний цикл повторяются n [эн] раз.

Переменная f [эф] определена как переменная логического типа. Логические переменные могут принимать одно из двух значений – «истина» или «ложь». Первоначально переменной f [эф] присвоено значение true [тру] – «истина». Предполагаем, что матрица симметрична.

Перебор элементов массива, расположенных по разные стороны от главной диагонали, производится во вложенных циклах. Если на каком-то шаге итерации элементы, расположенные по разные стороны от главной диагонали, не совпали, то логической переменной f [эф] присваивается значение false [фолс] – «ложь».

После завершения внутреннего цикла анализируется значение переменной f [эф]. Если значению этой переменной на каком-либо шаге итерации внутреннего цикла было присвоено значение false [фолс] – «ложь», то нет смысла продолжать проверку. Значит, матрица несимметрична, и внешний цикл прерывается оператором break [брэйк]. Вспомним, что оператор break [брэйк] прерывает только тот цикл, внутри которого он был описан, то есть прерывается внешний цикл.

Оптимизируем процесс вычислений. Обратите внимание на закон изменения параметра внутреннего цикла j [джи] – выделенное выражение на слайде. Этот факт позволяет сократить количество вычислений, то есть элементы верхнего треугольника сравниваются с элементами нижнего треугольника относительно главной диагонали.

Слайд 126

В результате вычислений на экране распечатается сообщение о симметричности или несимметричности матрицы.

На слайде приведен фрагмент, распечатывающий исходный массив в виде матрицы. Печать массива реализована во вложенных циклах.

Внешний цикл осуществляет переход от строки к строке.

Внутренний цикл выводит элементы одной отдельно взятой строки. Курсор удерживается на данной строке экрана, пока выводятся элементы одной строки матрицы. Между элементами используется табуляция, управляющий символ \t [обратный слеш ти], чтобы числа не сливались.

После вывода каждой строки осуществляется перевод курсора на начало новой строки. Для перевода курсора на начало следующей строки использована функция endl [эндэл].

Анализируется значение переменной f [эф]. И в зависимости от значения этой переменной, выдается сообщение о симметричности матрицы.

Слайд 127

На слайде приведен фрагмент, демонстрирующий решение задачи транспонирования матрицы. Фактически необходимо поменять местами элементы строк с элементами столбцов.

При решении этой задачи можно использовать подходы предыдущего примера, определения симметричности матрицы.

Для перебора всех элементов матрицы использованы вложенные циклы. Обратите внимание на подчеркнутый фрагмент в законе изменения параметра внутреннего цикла – j [джи]. Если в предыдущей задаче это действие только сокращает количество проверок, оптимизирует процесс вычислений, то в данном случае – это необходимо. Иначе матрица останется неизменной.

Фактически элементы верхнего треугольника, образованного главной диагональю, меняются местами с элементами нижнего треугольника квадратной матрицы. Индексы элементов верхнего и нижнего треугольника зеркально меняются.

Обмен выполняется с помощью функции swap [своп].

Слайд 128

Рассмотрим задачу. Требуется сформировать одномерный массив, каждый элемент которого равен произведению элементов соответствующего столбца двумерного массива, содержащего вещественные числа.

Согласно формулировке задачи, при описании массива необходимо выбрать тип float [флэут] или double [дабл] для элементов массива.

На слайде приведены конкретный пример исходного двумерного массива с именем a_array [эй э́ррей] и пример сформированного из исходного одномерного массива с именем b_array [би э́ррей].

Исходные данные – двумерный массив. Значения элементов двумерного массива необходимо ввести с клавиатуры.

Для наглядности полученных результатов распечатать исходный массив в виде матрицы. Под матрицей следует распечатать элементы одномерного массива, причем каждый элемент вывести на печать под соответствующим столбцом двумерного массива.

Слайд 129

Размер одномерного массива равен количеству столбцов в двумерном массиве.

Известно, что на языке С++ [си плюс плюс] описанные переменные автоматически не обнуляются. Более того, при подсчете произведения необходимо присвоить первоначально элементам одномерного массива значения, равные единице. В противном случае результат будет искажен.

Описаны два массива вещественного типа. Тип одномерного массива совпадает с типом элементов двумерного массива. Размер одномерного массива зависит от количества столбцов двумерного массива.

Размер двумерного массива, количество строк и количество столбцов, вводятся с клавиатуры. Обратите внимание на то, что массив описан после ввода его размера. Для тестирования и отладки программы можно задавать небольшие размеры массива, после чего можно использовать данную программу для обработки массивов большей размерности.

Ввод элементов двумерного массива осуществляется во вложенных циклах по строкам.

Обратите внимание на то, что переменные, которые являются параметрами цикла, индексами двумерного массива, описаны в структуре оператора цикла for [фо].

Следовательно эти переменные являются локальными и не доступны вне этого оператора. При повторном обращении к этим переменным их придется повторно описать. На следующем слайде продемонстрирован этот факт.

Индексы массива меняются с нуля. Напоминаем, что в массиве есть нулевая строка и нулевой столбец.

Слайд 130

На слайде – фрагмент программы, позволяющий вывести двумерный массив в виде матрицы. Вывод организован во вложенных циклах.

Во внутреннем цикле выводятся элементы одной отдельно взятой строки. Числа одной строки выводятся на одной строке экрана. Для того чтобы числа на слипались, используется управляющий символ \t [обратный слеш ти] – управляющий символ, табуляция. Во внутреннем цикле всего один оператор – вывод на печать, поэтому при построении тела массива в цикле фигурные скобки можно опустить.

Внешний цикл позволяет переходить от строки к строке. Во внешнем цикле два оператора:

- цикл вывода элементов одной строки;
- перевод курсора на начало следующей строки.

Поэтому тело внешнего цикла заключено в фигурные скобки.

Элементы одномерного массива надо вывести так, чтобы они четко располагались под соответствующим столбцом двумерного массива.

Слайд 131

На слайде продемонстрирован фрагмент формирования одномерного массива. Используется структура вложенных циклов.

Введены обозначения:

– единица – внутренний цикл;

– два – внешний цикл.

Во внутреннем цикле, параметром которого является индекс строки i [ай], подсчитывается произведение элементов одного отдельного столбца. Фактически формируется элемент одномерного массива.

Во внешнем цикле осуществляется переход от столбца к столбцу, и сформированное значение – элемент одномерного массива выводится на печать. В начале внешнего цикла перед формированием значения элемента одномерного массива ему присваивается значение равное единице. Если этого не сделать, не известно, какие значения будут присвоены элементам одномерного массива. Переменной, в которой накапливается произведение, необходимо предварительно присвоить значение равное единице.

При использовании вложенных циклов, мысленно соедините начало и конец циклов, и убедитесь, что они не пересекаются.

Слайд 132

Мы уже познакомились с понятием переменных – указателей. Доступ к элементам двумерного массива можно осуществит с помощью операций с указателями. Еще раз вспомним, что имя массива – указатель на начало области, отведенной под хранение элементов массива.

На слайде показаны две идентичные записи обращения к элементам двумерного массива. Обращаться к элементам двумерного массива можно и с помощью двух индексов, как показано в левом столбце. Причем первый индекс – всегда номер строки, а второй индекс – всегда номер столбца, на пересечении которых расположен данный элемент. Переменные, которые служат индексами массива, должны иметь целочисленный тип.

Идентичная запись обращения к элементам массива приведена в правом столбце. Ранее мы уже рассматривали переменные – указатели. Имя

массива – это фактически указатель на начало области, выделенной под хранение элементов массива. Наличие круглых скобок при обращении к элементам массива через указатели обусловлено синтаксисом языка C++ [си плюс плюс]. Внутренние скобки демонстрируют смещение, переход от строки к строке. Внешние скобки позволяют переходить от элемента к элементу в пределах одной строки. Наличие двух звездочек при обращении к элементам двумерного массива необходимо.

Далее приведены примеры обработки массивов с помощью указателей.

Слайд 133

Рассмотрим задачу. Требуется составить программу, которая выведет на экран элементы двумерного массива в порядке возрастания первого элемента каждой строки. Главное условие при решении данной задачи – исходный массив должен остаться неизменным.

Допустим, результаты сессии студентов содержатся в двумерном массиве. Каждая строка – это результаты сессии студента, соответствующие его номеру в списке группы. Список группы студентов отсортирован в алфавитном порядке. Требуется, не нарушая списка, вывести на печать результаты сессии в порядке возрастания успеваемости студентов. Вот ситуация, когда необходимо решить задачу, озвученную ранее. Исходный массив с результатами сессии должен остаться неизменным.

Для решения этой задачи будем использовать массив указателей. Размер массива указателей совпадает с количеством строк двумерного массива. Элементами массива указателей будут адреса первых элементов каждой строки. Сравнивать будем первые элементы каждой строки исходного массива, а менять местами будем значения элементов массива указателей, то есть адреса первых элементов. Фактически сортируются

элементы массива указателей. Сортировка будет зависеть от элементов исходного массива.

Печать двумерного массива осуществляется через массив указателей.

На слайде приведены примеры исходного массива и то, как должен выглядеть на экране результат. Показан также массив указателей.

Слайд 134

На слайде представлен фрагмент, в котором определены исходный двумерный массив и одномерный массив указателей. Размер одномерного массива указателей равен количеству строк исходного двумерного массива.

Напоминаем, указатели должны иметь тип информации, на которую они указывают. Следовательно, если исходный массив имеет тип double [дабл], то и тип массива указателей описан через тип double [дабл]. Символ *[звездочка] при описании определяет, что это массив указателей.

Во внутреннем цикле вводятся элементы массива по строкам. После выхода из внутреннего цикла адрес первого элемента каждой строки присваивается соответствующему элементу массива указателей. Адрес определяется с помощью уна́рной операции &[амперса́нт].

Обратите внимание на подчеркнутый и выделенный фрагмент – это и есть первый элемент в каждой строке, адрес которого присваивается соответствующему элементу массива указателей. Имя массива – указатель. Задавая только один индекс при обращении к двумерному массиву, мы фактически определяем адрес, по которому в памяти располагается первый элемент соответствующей строки. Ранее мы подробно рассматривали, что такое имя массива. Поэтому такая запись вполне допустима.

Слайд 135

Введем дополнительную переменную flag [флаг]. Она используется как признак при сортировке массива адресов, но не сами адреса анализируются, а анализируются элементы массива, расположенные по этим адресам. Переменная flag [флаг] имеет логический тип. Эта переменная может принимать одно из двух значений: «истина» или «ложь».

Чтобы получить доступ к числам, при сравнении используется символ *[звездочка], то есть сравниваются первые элементы строк исходного массива, а меняются местами адреса в массиве указателей.

Для обмена используется функция swap [своп], которая была рассмотрена ранее.

Сортировка реализуется во вложенных циклах. Внешний цикл, цикл с постусловием, продолжается до тех пор, пока значение переменной flag [флаг] – «ложно».

Условием прекращения внешнего цикла является значение переменной flag [флаг], равное истине, то есть имеет значение true [тру].

После того, как массив указателей отсортирован, цикл прекращается.

Слайд 136

Следующий фрагмент демонстрирует вывод двумерного массива через массив указателей.

При этом исходный массив остался неизменным.

Используя индексы i [ай] и j [джи], осуществляется переход от одного элемента к другому через указатели. Где индексы:

- j [джи] – определяет переход от столбца к столбцу, то есть переход от элемента к элементу одной строки;
- i [ай] – определяет переход от строки к строке.

Напомним, что элементы массива располагаются в памяти по строкам, независимо от того, как был организован ввод элементов исходного массива.

Переменная – указатель p [пэ] описана как указатель на информацию вещественного типа, так как элементы массива имеют вещественный тип. Поэтому увеличение значения переменной – указателя на единицу соответствует переходу к следующему элементу массива, что соответствует перемещению указателя на восемь байт. Так как под тип double [дабл] выделяется восемь байт памяти. Массив будет выведен на экран в виде матрицы.

На этом мы заканчиваем тему массивы. Однако с массивами нам придется иметь дело и при изучении следующих разделов нашего курса: при использовании функций, в теме «Структуры», в разделе «Символьные массивы».

Слайд 137

Тема 6. Функции (часть 1)

«С увеличением объема программы становится невозможным удерживать в памяти все детали. Естественным способом борьбы со сложностью любой задачи является ее разбиение на части. В C++ [си плюс плюс] задача может быть разделена на более простые и обозримые с помощью функций, после чего программу можно рассматривать в более укрупненном виде – на уровне взаимодействия функций» [5].

Функция – это основное понятие. Ранее говорилось о том, что каждая программа должна содержать единственную функцию с именем **main [мейн]**, которая называется главной функцией. Именно с этой функции начинаются вычисления в программе.

Помимо функции с именем **main [мейн]**, в программу может входить произвольное количество неглавных функций. Каждая функция должна быть определена или хотя бы описана до первого обращения к ней.

«Очень часто в программировании необходимо выполнять одни и те же действия. Например, мы хотим выводить пользователю сообщения об ошибке в разных местах программы, если он ввел неверное значение.

Функции – один из самых важных компонентов языка **C++ [си плюс плюс]**.

Любая функция имеет тип, так же как и любая переменная.

Функция может возвращать значение, тип которого в большинстве случаев аналогичен типу самой функции.

При объявлении функции после ее типа должно находиться имя функции и две круглые скобки – открывающая и закрывающая, внутри которых могут находиться один или несколько аргументов функции, которых также может не быть вообще.

После списка аргументов функции ставится открывающая фигурная скобка, после которой находится само тело функции.

В конце тела функции обязательно ставится закрывающая фигурная скобка. Функции очень сильно облегчают работу программисту и намного повышают читаемость и понятность кода, в том числе и для самого разработчика» [5].

Слайд 138

В определении функции указываются последовательность действий, выполняемых при ее вызове, называемая телом функции. Структура неглавных функций повторяет структуру главной функции.

Тело функции – блок или составной оператор, то есть последовательность описаний и операторов, – заключается в фигурные скобки.

В определении функции также указываются имя функции, тип функции и совокупность формальных параметров. Необходимо учесть, что тип функции определяет тип возвращаемого результата.

Функция возвращает только один результат. Если функция не возвращает никакого результата, то тип функции указывается с помощью служебного слова – **void [войд]**.

Имя функции – уникальный идентификатор, с помощью которого происходит вызов функции, обращение к функции.

«Функция – это самостоятельная единица программы, которая спроектирована для реализации конкретной подзадачи. Функция является подпрограммой, которая может содержаться в основной программе, а может быть создана отдельно, в библиотеке.

Каждая функция выполняет в программе определенные действия. Сигнатура функции определяет правила использования функции. Обычно сигнатура представляет собой описание функции, включающее имя функции, перечень формальных параметров с их типами и тип возвращаемого значения.

Семантика функции определяет способ реализации функции. Обычно представляет собой тело функции. После того как произошел вызов функции,

начинает работать данная функция. Если функцию нигде не вызвать, то этот код будет проигнорирован программой.

Переменные и константы, объявленные в разных функциях, независимы друг от друга, они даже могут иметь одинаковые имена» [5].

Слайд 139

Остановимся подробно на списке формальных параметров.

В списке формальных параметров обязательно задается их тип, то есть формальные параметры специфицируются.

На слайде приведены различные варианты спецификации формальных параметров. В первом варианте указывается тип и имя формального параметра. Во втором варианте указывается тип, имя формального параметра, определено значение формального параметра. В третьем варианте указан только тип формального параметра. Зарезервировано место в списке параметров, определена сигнатура функции. В дальнейшем можно внести изменения в функцию, ввести новую переменную, не изменяя вызывающей функции.

Совокупность формальных параметров определяет сигнатуру функции. Сигнатура функции зависит от количества формальных параметров, их типов и порядка следования.

Формальные параметры в теле функции не описываются, они используются при описании тела функции.

Список параметров может совсем отсутствовать. Но при этом для соблюдения синтаксиса пустые круглые скобки рядом с именем функции обязательны.

«Разбиение программ на функции дает следующие преимущества:

– функцию можно вызвать из различных мест программы, что позволяет избежать повторения программного кода;

– одну и ту же функцию можно использовать в разных программах;

– функции повышают уровень модульности программы и облегчают ее проектирование;

– использование функций облегчает чтение и понимание программы и ускоряет поиск и исправление ошибок.

С точки зрения вызывающей программы, функцию можно представить неким «черным ящиком», у которого есть несколько входов и один выход. С точки зрения вызывающей программы неважно, каким образом производится обработка информации внутри функции. Для корректного использования функции достаточно знать лишь ее сигнатуру» [2].

Слайд 140

Вызов функций осуществляется с указанием имени функции и списка фактических параметров. На слайде приведена структура обращения к функции.

При обращении к функции формальные параметры заменяются фактическими. Фактическими параметрами могут быть переменные или константы. Формальные и фактические параметры должны совпадать по типу и порядку следования.

На языке **C++ [си плюс плюс]** допускается переменное количество формальных параметров.

В списке формальных параметров сначала указываются имена и типы обязательных параметров, затем после запятой ставится многоточие – три точки – по требованию синтаксиса. Компилятору передается сообщение, что

дальнейших анализ количества и типов параметров не нужен. Переход от одного фактического параметра к другому осуществляется с помощью указателей.

Фактический аргумент – это величина, которая присваивается формальному аргументу при вызове функции. Таким образом, формальный аргумент – это переменная в вызываемой функции, а фактический аргумент – это конкретное значение, присвоенное этой переменной вызывающей функцией. Фактический аргумент может быть константой, переменной или выражением. Если фактический аргумент представлен в виде выражения, то его значение сначала вычисляется, а затем передается в вызываемую функцию. Если в функцию требуется передать несколько значений, то они записываются через запятую. При этом формальные параметры заменяются значениями фактических параметров в порядке их следования в сигнатуре функции.

Слайд 141

«Особенностью языка C++ является невозможность для функции определять внутри своего тела другую функцию. То есть не допускаются вложенные определения. Но вполне допускается вызов функции из другой функции.

Как говорилось выше, функция возвращает один результат в вызывающую функцию.

По окончании выполнения вызываемой функции осуществляется возврат значения в точку ее вызова. Это значение присваивается переменной, тип которой должен соответствовать типу возвращаемого значения функции. Функция может передать в вызывающую программу только одно значение» [1].

Важным оператором тела функции является оператор **return [ретен]**.

Указанное выражение в частном случае может быть просто именем переменной, которой в теле функции присвоено определенное значение.

Если функция не возвращает никакого результата, то выражение может отсутствовать.

Оператор **return [ретен]** определяет точку выхода из функции, возврат в вызывающую функцию. В теле функции допускается наличие нескольких операторов **return [ретен]**.

Этот оператор может присутствовать и в главной функции, тогда возвращаемое им значение анализируется операционной системой.

Слайд 142

«Часто бывает удобно, чтобы функции, реализующие один и тот же алгоритм для различных типов данных, имели одно и то же имя. Использование нескольких функций с одним и тем же именем, но с различными типами параметров, называется перегрузкой функций. Обычно имеет смысл давать разным функциям разные имена. Если же несколько функций выполняют одно и то же действие над объектами разных типов, то удобнее дать одинаковые имена всем этим функциям. Перегрузкой имени называется его использование для обозначения операций над разными типами. Собственно, уже для основных операций **C++ [си плюс плюс]** применяется перегрузка. Действительно: для операций сложения есть только одно имя – знак **+ [плюс]**, но оно используется для сложения и целых чисел, и чисел с плавающей точкой, и указателей. Такой подход легко можно распространить на операции, определенные пользователем, то есть на функции.

Для транслятора в таких перегруженных функциях общее только одно – имя. Очевидно, по смыслу такие функции сходны, но язык не способствует и не препятствует выделению перегруженных функций. Таким образом,

определение перегруженных функций служит, прежде всего, для удобства записи.

При вызове функции с именем **print [принт]** транслятор должен разобраться, какую именно функцию **print [принт]** следует вызывать. Для этого сравниваются типы фактических параметров, указанные в вызове, с типами формальных параметров всех описаний функций с именем **print [принт]**. В результате вызывается та функция, у которой формальные параметры наилучшим образом сопоставились с параметрами вызова. Или выдается ошибка, если такой функции не нашлось.

Если найдены два сопоставления по самому приоритетному правилу, то вызов считается неоднозначным, а значит, ошибочным. Эти правила сопоставления параметров работают с учетом правил преобразований числовых типов для языка **C++ [си плюс плюс]**» [3].

Слайд 143

«Итак, перегрузка функций – это возможность определять несколько функций с одним и тем же именем, но с разными параметрами.

На слайде приведены три примера. В первом примере выполняется операция вычитания с целыми числами. Однако, если нужно использовать числа с плавающей запятой, эта функция совсем не подходит, так как любые параметры типа **double [дабл]** будут конвертироваться в тип **int [инт]**. В результате будет теряться дробная часть значений.

Одним из способов решения этой проблемы является определение второй функций, пример второй на слайде.

Но есть и лучшее решение – перегрузка функции. Мы можем просто объявить еще одну функцию **subtract [субтракт]**, которая принимает параметры типа **double [дабл]**, третий пример на слайде.

Может показаться, что произойдет конфликт имен, но это не так. Компилятор может определить сам, какую версию функции **subtract [субтракт]** следует вызывать на основе аргументов, используемых в вызове функции. Если параметрами будут переменные типа **int [инт]**, то компилятор понимает, что мы хотим вызвать **subtract [субтракт]** с целыми параметрами. Если же мы предоставим два значения типа с плавающей запятой, то компилятор поймет, что мы хотим вызвать **subtract [субтракт]** с вещественными параметрами.

Правила обращения применяются в следующем порядке по убыванию их приоритета:

- точное сопоставление: сопоставление произошло без всяких преобразований типа или только с неизбежными преобразованиями;
- сопоставление с использованием стандартных целочисленных преобразований, то есть символьный тип **char [чар]** – в целый, тип **short [шорт]** – в целый, вещественный тип **float [флот]** – в тип **double [дабл]**;
- сопоставление с использованием стандартных преобразований, например, целых – в вещественные, беззнаковых – в целые» [3].

Слайд 144

Рассмотрим еще один пример использования перегрузки функций.

«Компилятор определяет, какую именно функцию требуется вызвать, по типу фактических параметров. Этот процесс называется разрешением перегрузки. Тип возвращаемого функцией значения в разрешении не

участвует. Механизм разрешения основан на достаточно сложном наборе правил, смысл которых сводится к тому, чтобы использовать функцию с наиболее подходящими аргументами и выдать сообщение, если такой не найдется» [2].

Допустим, имеется четыре варианта функции, определяющей наибольшее значение, как показано на слайде.

При вызове функции **max [макс]** компилятор выбирает соответствующий типу фактических параметров вариант функции. В приведенном примере будут последовательно вызваны все четыре варианта функции. Если точного соответствия не найдено, выполняются продвижения порядковых типов в соответствии с общими правилами.

Например, логические и символьные – в целые, вещественный тип **float [флот]** – в вещественный тип **double [дабл]**, целые – в вещественный тип **double [дабл]** или указателей – в **void [войд]**. Следующим шагом является выполнение преобразований типов. Если соответствие аргументов функций на одном и том же этапе может быть получено более чем одним способом, вызов считается неоднозначным и выдается сообщение об ошибке.

Неоднозначность может появиться при преобразовании типа, использовании параметров-ссылок, использовании аргументов по умолчанию.

Перегруженные функции должны находиться в одной области видимости, иначе произойдет сокрытие аналогично одинаковым именам переменных во вложенных блоках.

Слайд 145

Далее рассмотрим использование функций на конкретных примерах.

«Различают системные функции, входящие в состав систем программирования, и собственные функции. Системные функции хранятся в

стандартных библиотеках, и пользователю не нужно вдаваться в подробности их реализации. Достаточно знать лишь их сигнатуру. Примером системных функций, используемых ранее, являются функции **cout [си аут]** и **cin [син]**.

Собственные функции – это функции, написанные пользователем для решения конкретной подзадачи.

Возможны только две операции с функциями: вызов и взятие адреса. Указатель, полученный с помощью последней операции, можно впоследствии использовать для вызова функции.

Отметим, что формальные параметры в указателях на функцию описываются так же, как и в обычных функциях. При присваивании указателю на функцию требуется точное соответствие типа функции и типа присваиваемого значения» [5].

Итак, функция может возвращать результат в виде указателя. В этом случае и тип функции определяется как указатель.

На слайде приведен пример функции, которая возвращает результат в виде указателя. Поэтому при определении функции рядом с именем функции ставится символ звездочка.

Функция из двух строковых переменных вернет указатель на более короткую строку. Формальными параметрами являются имена массивов символов.

Так как информация, на которую ссылается указатель, имеет символьный тип, то и тип функции определен как символьный, то есть **char [чар]**.

В списке формальных параметров указаны имена двух массивов и пустые квадратные скобки. Размеры массивов можно не указывать. Сравнение продолжается до тех пор, пока в одной из строк не встретится

признак конца строки, символ **\0 [слеш ноль]**. Проверяется условие, и возвращается на более короткую строку указатель.

Слайд 146

На слайде представлена главная функция, вызывающая функцию с именем **min_str [мин строка]**, рассмотренную на предыдущем слайде.

Вывод результата осуществляется с помощью функция **puts [пут стринг]**, поэтому в программе указан заголовочный файл **stdio [эс ти дио]**.

Каждую строку можно было бы вводить как элементы одномерного массива. Но при работе с цепочкой символов удобнее рассматривать ее как строку. С клавиатуры поочередно вводятся две строки. При вводе цепочки символов с помощью функции **gets [гет стринг]** в конце этой строки автоматически сформируется признак конца строки. Именно по этому признаку, символу **\0 [слеш ноль]**, и выбирается, какая строка короче.

Если функция результата не возвращает, то обращение к ней может выглядеть как отдельный оператор. Если функция возвращает конкретный результат, то обращаться к ней необходимо в структуре какого-либо оператора или в структуре функции, как в нашем примере. Функция **min_str [мин стар]** является фактическим параметром функции вывода строки **puts [пут стринг]**. В данном случае функция **min_str [мин стар]** возвращает результат в виде указателя на ту строку, длина которой короче, и содержимое строки выводится на печать.

«Существуют функции, в описании которых невозможно указать число и типы всех допустимых параметров. Тогда список формальных параметров завершается многоточием, что означает – возможно еще несколько параметров. Такие функции пользуются для распознавания своих фактических параметров недоступной транслятору информацией» [5].

Слайд 147

Во многих языках программирования формальные и фактические параметры у функций должны совпадать по типу, порядку следования и количеству. На языке **C++ [си плюс плюс]** формальные и фактические параметры обязательно должны совпадать по типу и порядку следования, но допустимо переменное количество формальных параметров.

Для того чтобы разобраться с этой особенностью построения функций на языке **C++ [си плюс плюс]**, рассмотрим следующий пример. На слайде – пример программы с использованием функции с переменным количеством формальных параметров. В случае функции **summa [сумма]** первый параметр является целочисленным. Очевидно, что раз параметр не описан, транслятор не имеет сведений для контроля и стандартных преобразований типа этого параметра. При описании функции сначала указан явный параметр **k [кей]** целого типа, а затем – многоточие. Это означает, что при вызове этой функции можно указывать различное количество фактических параметров.

Данная функция подсчитывает сумму чисел. Количество чисел может быть различным.

Образуется сигнатура функции с именем «сумма», с переменной **k [кей]** во главе. Переменная **k [кей]** и определяет количество чисел.

Используя **& [амперсанд]**, операцию определения адреса, можно легко определить, что элементы массива расположены в памяти последовательно. То есть элементы нулевой, первый, второй и так далее размещены рядом, друг за другом. Мы уже знаем, что элементы массива расположены в памяти последовательно. Добавление единицы к указателю возвращает адрес памяти следующего объекта этого же типа данных.

Введена переменная – указатель **p [пи]** целого типа, благодаря которой и осуществляется переход от одного числа к другому. Первоначально этой

переменной присваивается адрес переменной **k [кей]**. Увеличение значения указателя на единицу соответствует перемещению на четыре байта – количество байт, отводимых под целые числа, то есть переходу к очередному числу. Данная функция возвращает сумму чисел.

«Чтобы обойти контроль типов параметров, лучше использовать перегрузку функций или стандартные значения параметров, чем параметры, типы которых не были описаны. Переменное количество формальных параметров становится необходимым только тогда, когда могут меняться не только типы, но и число параметров» [4].

Слайд 148

Хотя передача массива в функцию на первый взгляд выглядит так же, как передача обычной переменной, но **C++ [си плюс плюс]** обрабатывает массивы несколько иначе.

Когда обычная переменная передается по значению, то **C++ [си плюс плюс]** копирует значение аргумента в параметр функции. Поскольку параметр является копией, то изменение значения параметра не изменяет значение исходного аргумента.

Однако, поскольку копирование больших массивов – дело трудоёмкое, **C++ [си плюс плюс]** не копирует массив при его передаче в функцию. Вместо этого передается фактический массив. И здесь мы получаем побочный эффект, позволяющий функциям напрямую изменять значения элементов массива.

На слайде представлена главная функция, в которой дважды выполняется обращение к одной и той же функции вычисления суммы целых чисел. При первом обращении указано четыре фактических параметра. Первый параметр – это количество, а следующие три фактических параметра – это те числа, сумму которых надо подсчитать.

«Тип возвращаемого значения и типы параметров совместно определяют тип функции. Для вызова функции в простейшем случае нужно указать ее имя, за которым в круглых скобках через запятую перечисляются имена передаваемых аргументов. Вызов функции может находиться в любом месте программы, где по синтаксису допустимо выражение того типа, который формирует функция. Если тип возвращаемого функцией значения не **void** [войд], она может входить в состав выражений или, в частном случае, располагаться в правой части оператора присваивания» [3].

При втором вызове функции указано семь фактических параметров.

Первый параметр – количество, а затем перечислены сами числа.

Вызов функции осуществляется в структуре функции вывода на печать. После вывода суммы чисел первой последовательности указан управляющий символ табуляции, после чего выводится сумма второй последовательности чисел. Оба результата выводятся на одной строке экрана.

Ограничением этой функции является тот факт, что она применима только для подсчета суммы целых чисел. Так как переход от одного числа к другому осуществляется с помощью указателя, то и все фактические параметры должны иметь целый тип.

Результаты вычислений продемонстрированы на слайде.

Слайд 149

«Если тип возврата функции не **void [войд]**, то она должна возвращать значение указанного типа, использовав оператор **return [ретен]**. Исключением является функция **main [мейн]**, которая возвращает 0, если не предоставлено другое значение.

Когда процессор встречает в функции оператор **return [ретен]**, он немедленно выполняет возврат значения обратно. Любой код, который находится за оператором **return [ретен]** в функции, игнорируется.

Функция может возвращать только одно значение через оператор **return [ретен]** обратно. Это может быть либо число, либо значение переменной, либо выражение, у которого есть результат, либо определенное значение из набора возможных значений» [4].

На слайде представлена функция нахождения суммы элементов одномерного массива. Функция возвращает в главную функцию значение суммы элементов массива с помощью оператора **return [ретен]**.

Формальными параметрами являются размер массива и сам массив. Обратите внимание на пустые квадратные скобки при описании второго формального параметра – массива. Это вполне допустимо, фактически в функцию передается указатель на массив, адрес начала массива, адрес нулевого элемента.

Переменная **i [ай]** является индексом массива, она необходима для перебора элементов массива. В переменной **summa [сумма]** подсчитывается сумма элементов массива.

Обе переменные используются только в теле функции, являются локальными, описаны в теле функции. Время их существования – от начала и до конца работы функции. Вне этой функции эти переменные недоступны. Время их существования – от начала и до конца работы функции, в которой они описаны.

Обратите внимание, значению переменной **summa [сумма]** присвоено значение ноль в структуре оператора **for [фо]** во время инициализации. Как известно, инициализация выполняется только один раз в оператора цикла с параметром. В теме базовые алгоритмические структуры мы подробно рассматривали оператор цикла с параметром.

Слайд 150

На слайде представлен код главной функции.

Описан массив целых чисел. Размер массива и элементы массива вводятся с клавиатуры.

Вызов функции подсчета суммы элементов массива указан в структуре функции вывода на экран **cout [си аут]**. Возврат из функции осуществляется в ту же точку, из которой было обращение к функции и выводится значение суммы.

Обратите внимание на список фактических параметров. Указывается размер массива и имя массива при вызове функции.

«При использовании в качестве параметра массива в функцию передается указатель на его первый элемент, иными словами, массив всегда передается по адресу. При этом информация о количестве элементов массива теряется, и следует передавать его размерность через отдельный параметр» [5].

В функции как имя массива использовался формальный параметр **a [эй]**, а в главной функции – имя массива обозначено **array [эррей]**. Это вполне допустимо.

Еще раз напомним, что формальные и фактические параметры должны совпадать по типу и порядку следования.

При вызове функции формальные параметры заменяются на фактические параметры, именно с ними выполняются вычисления. Как известно, формальные параметры в теле функции не описываются, их тип определяется в списке формальных параметров при определении функции.

Слайд 151

В теме «Указатели и адреса объектов» говорилось о том, что допустимы указатели на функции. Рассмотрим пример обработки двумерного

массива с использованием функции, которая возвращает результат в виде указателя. На слайде приведена формулировка задачи. Приведен пример исходной и преобразованной матрицы с конкретными числами.

При решении этой задачи необходимо учесть следующее. Так как матрица квадратная, количество строк равно количеству столбцов. Число, определяющее размер матрицы, должно быть нечетным числом. Иначе на пересечении диагоналей нет элемента. В программе обязательно надо выполнить проверку на нечетность размера матрицы и, если условие нечетности не выполняется, выдать соответствующее сообщение. Вспомним свойство результативности алгоритма, которое гласит, что алгоритм должен приводить или к результату, или к сообщению о причинах его отсутствия.

В данной задаче результатом работы функции является номер строки и номер столбца, на пересечении которых находится максимальный элемент массива. Зная индексы максимального элемента, можно обменять его с элементом на пересечении главной и побочной диагоналей, индексы которого легко подсчитываются, через размер массива. Известно, что на языке **C++ [си плюс плюс]** функция возвращает только один результат. Поэтому результат возвращается в виде указателя на максимальный элемент массива.

Опишем массив как глобальную переменную, вне всяких функций. Тогда массив будет доступен во всех функциях, включая главную, и отпадает необходимость передавать массив в функции через формальные параметры.

Глобальные переменные существуют от начала и до конца работы программы. Не стоит злоупотреблять использованием глобальных переменных.

Слайд 152

«Все величины, описанные внутри функции, а также ее параметры, являются локальными. Областью их действия является функция. При вызове функции, как и при входе в любой блок, в стеке выделяется память под локальные автоматические переменные. Кроме того, в стеке сохраняется содержимое регистров процессора на момент, предшествующий вызову функции, и адрес возврата из функции для того, чтобы при выходе из нее можно было продолжить выполнение вызывающей функции. При выходе из функции соответствующий участок стека освобождается, поэтому значения локальных переменных между вызовами одной и той же функции не сохраняются.

Первая строка является описанием функции. Она задает функцию с параметром типа – целое, возвращающую значение типа – указатель. Описание функции необходимо для ее вызова. При вызове функции тип каждого фактического параметра сверяется с типом, указанным в описании функции, точно так же, как если бы инициализировалась переменная описанного типа. Это гарантирует надлежащую проверку и преобразования типов» [5].

На слайде представлена функция поиска максимального элемента в двумерном массиве целых чисел.

Звездочка перед именем при описании функции **max_array [макс эррей]** означает, что функция возвращает в виде результата указатель. Так как массив целых чисел, то и возвращаемый результат, то есть тип функции, – целого типа. В списке формальных параметров указан размер массива.

Обратите внимание на локальные переменные – это значение максимального элемента массива **max [макс]** и его индексы **imax [ай макс]** и **jmax [джей макс]**. Они определены в теле функции, так как в главной функции они не используются.

Необходимо определить номер строки и номер столбца, на пересечении которых находится максимальный элемент массива. Первоначально значению максимального присваивается значение элемента массива, расположенного на пересечении нулевой строки и нулевого столбца.

Поиск максимального элемента осуществляется во вложенных циклах.

Определив индексы максимального элемента с помощью унарной операции определения адреса **& [амперсанд]**, в главную функцию возвращается указатель на максимальный элемент массива, фактически адрес максимального элемента.

Слайд 153

На слайде представлено продолжение программы.

После ввода размера массива **n [эн]** необходимо проверить, является ли **n** [эн] нечетным числом. Проверка осуществляется в структуре логического оператора **if [иф]**. Вся последовательность операторов, в случае когда указанное условие истинно, то есть введено нечетное число, описана в фигурных скобках.

Если введено четное число, то есть указанное условие не выполняется, на экране появится сообщение об ошибке ввода. Это будет реализовано в следующем фрагменте программы, на следующем слайде.

Переменные **i [ай]** и **j [джей]**, используемые как индексы двумерного массива, описаны непосредственно в структуре оператора цикла. Это локальные переменные, доступные только в структуре этого оператора.

В программе определена переменная-указатель **p [пи]**, которой присваивается результат, возвращенный из функции **max_array [макс эррей]**.

Обращение к функции осуществляется в операторе присваивания. Напомним, что возврат из функции осуществляется в ту же точку, из которой происходил вызов функции. С помощью функции **swap [свап]** меняются местами максимальный элемент с элементом на пересечении главной и побочной диагоналей. Обмен осуществляется через указатель на максимальный элемент.

Когда перед указателем **p [пи]** ставится знак звездочка, осуществляется обращение к информации по заданному адресу, фактически к максимальному элементу.

Слайд 154

На слайде представлен фрагмент вывода преобразованного массива в виде матрицы.

Вывод осуществляется во вложенных циклах. Внутренний цикл позволяет вывести на экран элементы одной отдельно взятой строки. Внешний цикл позволяет перейти от строки к строке.

Обратите внимание: переменные, которые служат индексами двумерного массива, описаны в структуре операторов цикла.

Выделена часть оператора проверки условия – структура **else [елс]**.

Дополнительные фигурные скобки определяют последовательность действий, которые будут пройдены при выполнении условия задачи, то есть если введено нечетное число. В противном случае, то есть если введено четное число, на экране появится сообщение о некорректном вводе.

Сделаем вывод: если необходимо вернуть из функции несколько результатов, можно воспользоваться указателями или глобальными переменными.

Например, в нашей задаче индексы максимального элемента массива можно было описать вне всяких функций, тогда они были бы доступны во всех функциях, включая главную. Отпадает необходимость в возврате значений из функции. Но не стоит злоупотреблять глобальными переменными, значения которых сохраняются от начала и до конца работы программы.

Слайд 155

Тема 6. Функции (часть 2)

Рассмотрим задачу. Нужно составить программу поиска некоторого заданного числа **k [кей]** в массиве целых чисел. Определить местоположение найденного числа в массиве. Использовать одну функцию как для поиска в одномерном массиве, так и для поиска в двумерном массиве. На слайде приведены введенные обозначения.

Составим универсальную функцию, к которой будем обращаться для поиска определенного числа как в одномерном массиве, так и в двумерном массиве. Единственным ограничением в данной задаче является тип элементов массива. Массив должен содержать целые числа. Это связано с тем, что переход от одного элемента массива к другому, как в одномерном, так и в двумерном массивах, будет осуществляться через указатель. Указатель также должен иметь целый тип.

Увеличение значения указателя, то есть адреса, на единицу фактически означает переход от одного элемента массива к другому элементу. Вспомним, что элементы двумерного массива хранятся в памяти по строкам, и с помощью указателя можно перебрать все элементы и двумерного массива.

Введены дополнительные переменные **m [эм], k [кей]** и **p [пи]**. В одномерном массиве местоположение элемента определяется одним индексом. В двумерном массиве каждый элемент определяется двумя индексами. Функция может вернуть только один результат. Так как эта функция будет использоваться и для поиска в одномерном массиве, и для поиска в двумерном массиве, результат возвращается в виде указателя на найденный элемент в массиве. При обращении к этой функции для поиска числа в двумерном массиве необходимо указать количество элементов, то есть произведение количества строк на количество столбцов.

Через этот указатель будем определять местоположение найденного элемента как в одномерном массиве, так и в двумерном массиве.

Слайд 156

На слайде представлена функция поиска числа в массиве, которая возвращает результат в виде указателя на найденный в массиве элемент.

Первая часть определения функции задает ее имя, тип возвращаемого значения, а также типы и имена формальных параметров. Значение возвращается из функции с помощью оператора **return [ретен]**.

Соответственно, перед именем функции ставится символ звездочка. Тип функции целый, определяется типом элементов массива, типом возвращаемого результата.

В списке формальных параметров:

– указатель на начало массива, то есть имя массива;

– количество элементов в массиве, одномерном или двумерном;
– искомое число **k [кей]**.

Имя массива является указателем на начало массива. Увеличение значения переменной **a [эй]** на единицу, то есть увеличения адреса на единицу, при каждом прохождении через цикл соответствует переходу от одного элемента к другому.

Напомним, что элементы двумерного массива расположены в памяти по строкам. Перебрав элементы одной строки, указатель переходит к перебору элементов следующей строки.

Через этот указатель определяется местоположение числа в массиве, как одномерном, так и в двумерном. Именно через указатель будем определять местоположение элемента.

Если число в массиве на найдено, функция вернет с помощью оператора **return [ретен]** константу **NULL [нул]**.

Слайд 157

Рассмотрим фрагмент использования описанной выше функции для поиска искомого числа в двумерном массиве.

Значения элементов массива, как одномерного, так и двумерного, определены в программе. Значение искомого числа вводится с клавиатуры.

При первом обращении к функции **poisk [поиск]** фактическими параметрами являются:

– имя двумерного массива;
– константа, определяющая количество элементов двумерного массива;
– искомое число.

Введены две переменные: **S [эс]** – номер строки и **St [эс ти]** – номер столбца. С помощью этих переменных по указанным формулам через указатель, полученный как результат работы функции, определяются номер

строки и номер столбца, на пересечении которых находится найденный элемент. Двумерный массив инициализирован во время описания конкретными значениями. Обратите внимание, что в формулах определения номера строки и номера столбца присутствует число три. Это количество столбцов двумерного массива. Если размерность массива другая, в формулу необходимо внести изменения.

Обращение к функции **poisk [поиск]** происходит в теле функции вывода **cout [си аут]**. В функции вывода предусмотрен формат вывода найденного элемента с указанием номера строки и номера столбца, на пересечении которых расположен найденный элемент. На экран выведется элемент с указанием его положения в двумерном массиве.

Если элемент не найден в двумерном массиве, выдается соответствующее сообщение. Для этого используется константа **NULL [нул]**.

Слайд 158

Следующий фрагмент демонстрирует обращение к той же функции **poisk [поиск]** для нахождения искомого числа в одномерном массиве. Одномерный массив инициализирован конкретными значениями при объявлении.

Фактическими параметрами при обращении к функции являются имя одномерного массива, количество элементов в одномерном массиве и искомое число.

Функция вернет указатель на найденный элемент в одномерном массиве. Если число будет найдено в массиве, то на экран выведется элемент одномерного массива с указанием индекса, значение которого подсчитывается через указатель. Индекс элемента определяется как разность между указателем, то есть адресом найденного элемента, и адресом начала массива. Индекс определяет положение элемента в общем наборе, массиве.

Если число в массиве не найдено, то есть функция возвращает константу **NULL [нул]**, появится соответствующее сообщение.

Если значения массивов не определять в явном виде в программе, то необходимо ввести дополнительные переменные как количество элементов в одномерном и двумерном массивах соответственно. И тогда можно использовать ту же функцию поиска.

На следующем слайде будет рассмотрена работа этой функции поиска на конкретных примерах.

Разделение программы на функции позволяет избежать избыточности кода, поскольку функцию записывают один раз, а вызывать ее на выполнение можно многократно из разных точек программы. Процесс отладки программы, содержащей функции, можно лучше структурировать. Часто используемые функции можно помещать в библиотеки. Таким образом создаются более простые в отладке и сопровождении программы.

Слайд 159

На слайде приведены два варианта результатов при вводе различных значений переменной **k [кей]**.

Переменная, значение которой ищется в массивах, вводится с клавиатуры. Для двумерного массива это общее количество элементов массива.

В левой колонке – результаты работы функции. В правой колонке – значения элементов одномерного и двумерного массивов.

В первом случае, при значении искомой переменной, равной двум, число будет найдено и в двумерном, и в одномерном массивах. Положение найденного элемента в массивах определено с помощью индексов, указанных в квадратных скобках.

Во втором варианте, при значении искомой переменной, равной десяти, число в одномерном массиве найдено, а в двумерном массиве число не найдено, о чем выдается сообщение.

«Так же часто бывает полезен массив указателей на функции. Например, можно реализовать систему меню для редактора с вводом, управляемым мышью, используя массив указателей на функции, реализующие команды.

Но как быть, когда нам потребуется вызвать функцию, которая должна получить несколько значений в результате вычислений. Здесь для этого проще всего использовать так называемые функции типа **void [войд]**. Такие функции не содержат оператора **return [ретен]** и поэтому как бы ничего не передают наружу. Однако в **C++ [си плюс плюс]** существует понятие глобальной переменной, то есть такой переменной, которая доступна в любой функции данной программы. Если обратится к функции типа **void [войд]** просто по имени, без использования присваивания, и внутри нее пересчитать глобальные переменные, то после этого в основной программе значения глобальных переменных изменятся» [3].

Напомним, что глобальные переменные описываются до открытия главной функции **main [мейн]**.

Слайд 160

«Рекурсивная функция – это функция, которая вызывает саму себя. Это в случае прямой рекурсии. Существует и косвенная рекурсия, когда две или более функций вызывают друг друга. Когда функция вызывает себя, в стеке создаётся копия значений её параметров, после чего управление передаётся первому исполняемому оператору функции. При повторном вызове процесс повторяется.

Рекурсивные функции являются альтернативой циклам. Рекурсивные функции в основном используются для упрощения проектирования алгоритмов. В программировании рекурсия тесно связана с функциями, точнее, именно благодаря функциям в программировании существует такое понятие, как рекурсия или рекурсивная функция. Простыми словами, рекурсия – определение части функции через саму себя, то есть это функция, которая вызывает саму себя. Типичными рекурсивными задачами являются задачи нахождения **n! [эн факториал]**, числа Фибоначчи. Такие задачи мы уже решали, но с использованием циклов, то есть итеративно» [3].

На слайде представлена функция вычисления факториала, произведения натуральных чисел от одного до **n** [эн].

Для вычисления произведения используется цикл с параметром. Однако есть очень красивый подход замены циклов рекурсией.

Рекурсия – фундаментальное понятие в математике и компьютерных науках. В программировании рекурсивной называется функция, которая обращается сама к себе.

Рекурсивная функция не может вызывать себя до бесконечности, следовательно, важная особенность рекурсивной функции – наличие условия завершения, позволяющее функции прекратить вызывать себя.

При выполнении рекурсивной функции сначала происходит рекурсивное погружение, а затем возврат вычисленных результатов.

Рекурсивную функцию всегда можно преобразовать в не рекурсивную, итеративную, использующую циклы. И эта функция выполняла бы те же вычисления. И наоборот, используя рекурсию, можно реализовать любое вычисление, предполагающее использование циклов, не прибегая к циклам.

Слайд 161

«Вообще говоря, всё то, что решается итеративно, можно решить рекурсивно, то есть с использованием рекурсивной функции. Всё решение сводится к решению основного или, как ещё его называют, базового случая. Существует такое понятие, как шаг рекурсии, или рекурсивный вызов. В случае, когда рекурсивная функция вызывается для решения сложной задачи, выполняется некоторое количество рекурсивных вызовов, или шагов, с целью сведения задачи к более простой.

Одной из сложных для понимания тем в большом числе случаев оказывается тема рекурсии. Даже шутка есть: чтобы понять рекурсию, надо сначала понять рекурсию.

В программировании рекурсией называют зацикливание функций путём вызовов функций вызовами функций.

Различают прямую рекурсию и косвенную рекурсию.

Прямая – это прямой вызов из некоторой функции этой же самой функции.

Косвенная – это вызов из первой функции второй функции, когда вторая функция в своём вычислении использует первую функцию» [3].

На слайде представлена рекурсивная функция вычисления факториала с именем **fact [факт]**. В теле функции введена переменная **f [эф]** для накапливания произведения чисел, которой первоначально необходимо присвоить значение единица. Переменная является локальной, время ее существования – от начала и до конца работы этой функции.

Функция многократно обращается сама к себе, каждый раз уменьшая значение формального параметра **n [эн]** на единицу. На слайде это выделенный и подчеркнутый фрагмент.

Погружение в функцию продолжается до тех пор, пока значение параметра **n [эн]** не станет равно нулю. После этого происходит

многократный выход из рекурсивной функции, и при каждом выходе подсчитывается произведение.

Функция возвращает вычисленное произведение в вызывающую функцию с помощью оператора **return [ретен]**.

Слайд 162

«Рекурсивные вызовы функций работают точно так же, как и обычные вызовы функций. Однако программа, приведенная выше, иллюстрирует наиболее важное отличие простых функций от рекурсивных: вы должны указать условие завершения рекурсии. В противном случае функция будет выполняться бесконечно, фактически до тех пор, пока не закончится память в стеке вызовов.

Условие завершения рекурсии – это условие, которое при его выполнении остановит вызов рекурсивной функции самой себя.

Вызов функции влечет за собой некоторые дополнительные накладные расходы, связанные с передачей управления и аргументов в функцию, а также возвратом вычисленного значения. Поэтому итерационная процедура вычисления факториала будет несколько более быстрым решением. Чаще всего итерационные решения работают быстрее рекурсивных. Любые рекурсивные процедуры и функции, содержащие всего один рекурсивный вызов самих себя, легко заменяются итерационными циклами.

Еще одним недостатком рекурсии является то, что ей может не хватать для работы стека. При каждом рекурсивном вызове в стеке сохраняется адрес возврата и передаваемые аргументы. Если рекурсивных вызовов слишком много, отведенный объем стека может быть превышен» [3].

Для того чтобы представить изменения параметров при каждом обращении к рекурсивной функции и значения, с которыми происходит выход, рассмотрим конкретный пример вычисления факториала числа пять.

На слайде поэтапно рассмотрены значения, с которыми происходит погружение в рекурсивную функцию. Это продемонстрировано в первой колонке.

Функция вычисления факториала имеет один формальный параметр. При каждом повторном обращении к функции значение фактического параметра уменьшается на единицу. Во второй колонке показан возврат значений рекурсивной функции на каждом шаге итерации. Возвращаемые результаты функции – фактически и есть произведение чисел.

Слайд 163

«Автор функции решает, что означает её возвращаемое значение. Некоторые функции используют возвращаемые значения в качестве кодов состояния для указания результата выполнения функции – успешно ли выполнение или нет. Другие функции возвращают определенное значение из набора возможных значений. Кроме того, существуют функции, которые вообще ничего не возвращают» [4].

Рассмотрим задачу. Нужно определить, является ли заданное целое число **n [эн]** точной степенью двойки. На слайде представлена рекурсивная функция, реализующая задачу. Формальным параметром функции является исходное число.

Функция многократно обращается сама к себе, подавая в виде формального параметра число, деленное на два. Этот фрагмент выделен на слайде.

Как известно, в рекурсивных функциях обязательно должно присутствовать условие прекращения рекурсии. В данной задаче процесс так

называемого погружения продолжается до тех пор, пока в результате деления числа на два результат не станет равен единице. Это значит, что число является точной степенью двойки. Вторым условием прекращения рекурсии является условие, когда в результате деления числа на два результат – дробное число.

Функция возвращает одно из двух значений: единица, если число является точной степенью двойки, и ноль – в противном случае.

При реализации этого алгоритма можно определить эту степень, если число является точной степенью двойки. В функции присутствует еще одна переменная **f [эф]**, которая подсчитывает количество погружений, фактически определяет степень двойки.

Так как функция может вернуть только один результат, и это признак того, является ли число точной степенью двойки, переменная, определяющая степень двойки, описана как глобальная переменная. Соответственно, значение переменной **f [эф]** можно использовать в главной функции.

Слайд 164

В главной функции анализируется значение введенного числа, и если оно отрицательное, в результате выполнения программы на печать выдается сообщение, что введенное число недопустимо, и вычисления прекращаются.

Если функция не возвращает никакого результата, то обращение к ней можно описать в программе в отдельном выражении, как отдельный оператор. Данная функция возвращает результат, поэтому обращение к рекурсивной функции происходит в структуре оператора проверки условия **if [иф]**.

Если введенное число является точной степенью двойки, функция возвращает значение единица, в противном случае – функция возвращает значение ноль.

Если рекурсивная функция возвращает в виде результата единицу, на экране выдается, какая это степень двойки.

Для определения степени двойки используется глобальная переменная **f [эф]**, значение которой подсчитывалось в рекурсивной функции. При каждом обращении к рекурсивной функции значение переменной увеличивается на единицу. Переменная **f [эф]** описана и определена вне всяких функций.

Если рекурсивная функция вернет значение, равное нулю, появится соответствующее сообщение, что введенное число не является точной степенью двойки.

«Глобальные переменные видны во всех функциях, где не описаны локальные переменные с теми же именами, поэтому использовать их для передачи данных между функциями очень легко. Тем не менее это не рекомендуется, поскольку затрудняет отладку программы и препятствует помещению функций в библиотеки общего пользования. Нужно стремиться к тому, чтобы функции были максимально независимы, а их интерфейс полностью определялся прототипом функции» [5].

Слайд 165

Функция должна быть определена до первого обращения к ней или хотя бы описана.

На слайде показано описание функции. Если функция описана ниже вызывающей, необходимо указать прототип функции. Указывается тип функции, имя функции и список формальных параметров. Обратите

внимание, что описание функции завершается точкой с запятой после круглой скобки. Тело функции отсутствует.

«Поскольку прототип функции является стейтментом, он также заканчивается точкой с запятой. Определение необходимо для корректной работы компоновщика, линкера. Если вы используете идентификатор без его определения, то линкер выдаст ошибку.

В языке C++ [си плюс плюс] есть правило, которое состоит из трех частей:

– внутри файла функция, объект, тип могут иметь только одно определение;

– внутри программы объект или обычная функция могут иметь только одно определение;

– внутри программы типы, шаблоны функций и встроенные функции могут иметь несколько определений, если они идентичны.

Нарушение первой части правила приведет к ошибке компиляции. Нарушение второй или третьей частей правила приведет к ошибке линкинга, компоновки.

Объявление – это стейтмент, который сообщает компилятору о существовании идентификатора и о его типе. Объявление – это всё, что необходимо для корректной работы компилятора, но недостаточно для корректной работы линкера. Определение обеспечит корректную работу как компилятора, так и линкера» [1].

Определение функции, тело функции могут быть указаны после вызывающей функции.

Допустима вложенность функций, когда одна функция, не главная, вызывает другую. При построении таких функций придерживаются правил, изложенных выше.

Функция может быть определена и в одном файле с вызывающей функцией, и в разных файлах.

Подключение функции к файлу происходит с помощью команды препроцессора, как и библиотечных, стандартных функций.

Слайд 166

Где, когда и как эффективно использовать функции – одна из наиболее распространенных проблем.

На слайде представлена формулировка задачи, при решении которой будут использованы прототипы функций. Если функция определена ниже вызывающей, необходимо в теле вызывающей функции использовать прототипы функций. На примере следующей задачи подробно рассмотрим, как в программе используются прототипы функций.

Предполагается построение двух функций.

Первая функция реализует вывод двумерного массива в виде матрицы. К этой функции в программе будем обращаться два раза: для вывода на экран исходного массива и для вывода преобразованного массива.

Вторая функция сортирует элементы каждой строки в порядке возрастания.

«В определении, в объявлении и при вызове одной и той же функции типы и порядок следования параметров должны совпадать. На имена параметров ограничений по соответствию не накладывается, поскольку функцию можно вызывать с различными аргументами. В прототипах имена компилятором игнорируются. Они служат только для улучшения читаемости программы» [3].

Формальными параметрами первой функции являются размер массива и имя массива.

Формальным параметром второй функции является номер строки, элементы которой необходимо отсортировать.

Обе функции будут определены ниже главной функции, поэтому в теле главной функции необходимо указать прототипы перечисленных выше функций.

Слайд 167

«Объявление функции предполагает определение таких понятий, как прототип, заголовок, сигнатура. При определении функции задается ее имя, тип возвращаемого значения и список передаваемых параметров. Определение функции содержит, кроме объявления, тело функции, представляющее собой последовательность операторов и описаний в фигурных скобках» [5].

Функции, используемые в программе, описаны ниже обращения к этим функциям. Поэтому в главной функции указаны прототипы трех функций:

– **input_mas [инпут мас]** – предназначена для ввода элементов матрицы;

– **print_mas [принт мас]** – используется для вывода на печать массива в виде матрицы;

– **tran_mas [тран мас]** – выполняет транспонирование исходной матрицы.

На слайде представлен код функции **main [мейн]**. Обратите внимание, описание прототипа завершается точкой с запятой.

В списке формальных параметров прототипов функций размер массива обозначен только типом. При описании прототипа каждой функции указано имя функции и список формальных параметров. Это было необходимо для создания сигнатуры функции. Тип функций, то есть тип возвращаемого результата, указан **void [войд]**, поэтому обращение к этим функциям представлено как отдельный оператор в программе.

Опишем последовательность действий, выполняемых программой:

– обращение к функции **input_mas [иннут мас]** для ввода исходной матрицы;

– функция **print_mas [принт мас]** выведет на печать исходную матрицу;

– функция **tran_mas [тран мас]** транспонирует исходную матрицу;

– повторное обращение к функции **print_mas [принт мас]** выведет на печать транспонированную матрицу.

Функции, к которым обращаются из главной функции, будут описаны ниже главной. Они представлены на следующих слайдах.

Слайд 168

«Естественным способом борьбы со сложностью любой задачи является ее разбиение на части. В **C++ [си плюс плюс]** задача может быть разделена на более простые и обозримые с помощью функций, после чего программу можно рассматривать в более укрупненном виде – на уровне взаимодействия функций. Это важно, поскольку человек способен помнить ограниченное количество фактов» [5].

На данном слайде описана функция ввода элементов массива по строкам. Ввод осуществляется во вложенных циклах.

«При вызове функции выделяется память под её формальные параметры, и каждому формальному параметру присваивается значение соответствующего фактического параметра. Семантика передачи параметров идентична семантике инициализации. В частности, проверяется соответствие типов формальных и фактических параметров и при необходимости выполняются либо стандартные, либо определённые пользователем преобразования типов. Существуют специальные правила для передачи массивов в качестве параметров» [3].

Тело функции указано в фигурных скобках. При обращении к этой функции из главной передается размер массива и указатель на первый элемент матрицы, элемент нулевой строки и нулевого столбца.

Как известно, имя массива – это указатель на начало области, выделенной под хранение элементов массива.

Вспомните, что при описании прототипа функции переменная, определяющая размер массива, не была указана. Определен только тип формального параметра. При обращении к этой функции из главной указывается переменная, определенная как размер матрицы. Так как матрица квадратная, количество строк равно количеству столбцов.

Ввод элементов осуществляется по строкам. Это сообщение предусмотрено в теле функции.

Слайд 169

«Все величины, описанные внутри функции, а также ее параметры, являются локальными. Областью их действия является функция. При вызове функции, как и при входе в любой блок, в стеке выделяется память под локальные автоматические переменные. Кроме того, в стеке сохраняется содержимое регистров процессора на момент, предшествующий вызову функции, и адрес возврата из функции для того, чтобы при выходе из нее можно было продолжить выполнение вызывающей функции. При выходе из функции соответствующий участок стека освобождается, поэтому значения локальных переменных между вызовами одной и той же функции не сохраняются.

Напомним, что в определении, в объявлении и при вызове одной и той же функции типы и порядок следования параметров должны совпадать. На имена параметров ограничений по соответствию не накладывается,

поскольку функцию можно вызывать с различными аргументами, а в прототипах имена компилятором игнорируются» [2].

На слайде продемонстрирована функция вывода на экран двумерного массива в виде матрицы. К этой функции мы обращаемся дважды: для вывода на печать исходной матрицы и транспонированной матрицы.

Вывод массива в виде матрицы осуществлен во вложенных циклах.

Структура вложенных циклов схематично показана на слайде.

Внутренний цикл выводит на печать элементы одной, отдельно взятой строки. На слайде обозначен цифрой один.

Внешний цикл осуществляет переход от строки к строке матрицы, на слайде он обозначен цифрой два. В конце каждой строки предусмотрен перевод курсора на начало следующей строки с помощью управляющего символа **\n [слеш эн]**.

При определении функции указаны формальные параметры – размер массива и имя массива.

Обратите внимание, функция не возвращает результата, поэтому тип функции при описании указан с помощью служебного слова **void [войд]**.

Слайд 170

На слайде представлена функция транспонирования квадратной матрицы. Ранее мы рассматривали алгоритм транспонирования матрицы. Элементы верхнего треугольника и элементы нижнего треугольника, образованные главной диагональю, меняются местами.

Обратите внимание на закон изменения параметра внутреннего цикла **j [джей]**. Именно это выражение позволяет транспонировать матрицу. Этот фрагмент подчеркнут на слайде.

Рассмотренная задача продемонстрировала возможность описывать функции ниже вызывающей, при этом нельзя забывать о прототипе функции.

Использование функций упрощает понимание кода. Фактически перечислена последовательность действий.

«Использование функций является первым шагом к повышению степени абстракции программы и ведет к упрощению ее структуры. Разделение программы на функции позволяет также избежать избыточности кода, поскольку функцию записывают один раз, а вызывать ее на выполнение можно многократно из разных точек программы. Процесс отладки программы, содержащей функции, можно лучше структурировать. Часто используемые функции можно помещать в библиотеки. Таким образом создаются более простые в отладке и сопровождении программы.

Следующим шагом в повышении уровня абстракции программы является группировка функций и связанных с ними данных в отдельные файлы, компилируемые раздельно. Получившиеся в результате компиляции объектные модули объединяются в исполняемую программу с помощью компоновщика. Разбиение на модули уменьшает время перекомпиляции и облегчает процесс отладки, позволяя отлаживать программу по частям. Это уменьшает общий объем информации, которую необходимо одновременно помнить при отладке. Разделение программы на максимально обособленные части является сложной задачей, которая должна решаться на этапе проектирования программы» [5].

На этом мы завершаем изучение темы «Функции».

Слайд 171

Тема 7. Массивы символов. Строки (часть 1)

«Символ – элементарная единица, некоторый набор которых несет определенный смысл. В языке программирования **C++ [си плюс плюс]** предусмотрено использование символьных литерал. Символьный литерал – это целочисленное значение типа **int [инт]**, представленное в виде символа,

заключённого в одинарные кавычки. В таблице **ASCII [аски]** кодов представлены символы и их целочисленные значения» [3].

На слайде приведены два примера, два способа описания и определения строки символов.

Работа со строками реализуется путем использования одномерных массивов символов типа **char [чар]**.

Рассмотрим первый пример. На языке **C++ [си плюс плюс]** допускаются строковые константы – это строки символов, заключенные в двойные кавычки.

Любая последовательность символов, заключенная в «» [двойные кавычки], рассматривается как строковая константа. Символьная строка – это одномерный массив, заканчивающийся нулевым байтом. Для нулевого байта определена специальная символьная константа – **\0 [слеш ноль]**, целочисленное значение которой равно 0. Если строка должна содержать **n [эн]** символов, то при описании массива необходимо задать размер массива, равный **n + 1 [эн плюс один]**. В конце строковой константы автоматически ставится символьная константа **\0 [слеш ноль]**. Так, последовательность символов, представляющая собой строковую константу, будет размещена в оперативной памяти компьютера, включая нулевой байт. Строковые константы размещаются в статической памяти. Начальный адрес последовательности символов в двойных кавычках трактуется как адрес строки. Строковые константы часто используются для осуществления диалога с пользователем.

Рассмотрим второй пример. Строка символов представляет собой одномерный массив символов типа **char [чар]**. Размер массива определяется количеством символов в массиве. При разной организации символьного массива и подходы обработки массивов будут разными.

Слайд 172

На языке **C++ [си плюс плюс]** существуют функции, позволяющие обрабатывать строку символов. Рассмотрим эти функции и примеры их использования.

В таблице указаны заголовочные файлы, которые необходимо подключать к программе для использования функций обработки строк символов. Название образовано от имени строчного типа данных **string [стринг]**, от английского слова «строка». В языке **C++ [си плюс плюс]** и его предшественнике – языке программирования **C [си]** нет встроенной поддержки строкового типа данных, вместо этого используется массив символов. Являясь частью стандартной библиотеки **C++ [си плюс плюс]**, эти объекты также являются частью стандартного пространства имён – **std [эс ти ди]**.

Функция **getch [гетч] –** ожидается ввод любого символа. Пустые скобки рядом с именем этой функции необходимы для соблюдения синтаксиса языка. Эта функции используется, как правило, в конце программы, чтобы успеть проанализировать результаты перед возвратом к программному коду.

При использовании этой функции необходимо подключить к программе заголовочный файл **conio [конио]**.

Функция **gets [гет стринг]** используется для ввода любой цепочки символов. Автоматически в конце строки ставится нулевой байт.

Функция **puts [пут стринг]** используется для вывода строки символов до нулевого символа, то есть символа **\0 [слеш ноль]**.

В качестве параметра этих функций ставится имя массива символов.

При использовании этих функций необходимо подключить к программе заголовочный файл **stdio [эс ти дио]**.

Слайд 173

«Есть много случаев, когда мы не знаем заранее, насколько длинной будет наша строка. Например, рассмотрим следующую проблему. При написании программы мы просим пользователя ввести свое имя. Насколько длинным оно будет? Это неизвестно до тех пор, пока пользователь его не введет.

В таком случае мы можем объявить массив размером больше, чем нам нужно.

Тем не менее использование функции **strcpy [стринг копи]** может легко вызвать переполнение массива, если не быть осторожным!» [3].

Рассмотрим функцию копирования одной строки в другую.

Для использования этой функции необходимо подключить к программе заголовочный файл **string [стринг]**.

Функция копирует строку **s1 [эс один]** в строку **s [эс]**.

Во второй строке таблицы показана структура этой функции в общем виде, которая разъясняет работу этой функции.

Из описания структуры видно – результатом работы этой функции является указатель. В скобках указываются два параметра:

- строка **s [эс]**, в которую копируется символы;
- строка **s1 [эс один]**, которая копируется.

Строка **s [эс]** должна быть достаточно большой, чтобы в нее поместилась строка **s1 [эс один]**. Если места мало, компилятор не выдаст ошибки, но может привести к порче других данных.

На слайде показан пример использования этой функции на конкретном примере. Обратите внимание на то, как происходит вызов этой функции – в структуре функции вывода на печать.

Слайд 174

«Большинство операций языка **Си [си]**, имеющих дело со строками, работает с указателями. Для размещения в оперативной памяти строки символов необходимо:

– выделить блок оперативной памяти под массив;

– проинициализировать строку.

Для выделения памяти под хранение строки могут использоваться функции динамического выделения памяти. При этом необходимо учитывать требуемый размер строки. Под хранение строки выделяются последовательно идущие ячейки оперативной памяти. Таким образом, строка представляет собой массив символов. Для хранения кода каждого символа строки отводится один байт» [3].

Результатом работы многих функций обработки строк, массивов символов является указатель. Следующая функция **strncpy [стринг эн копи]** копирует, но только часть строки **s1 [эс один]** в строку **s [эс]**. Функция также возвращает указатель на начало строки **s [эс]**.

У данной функции три параметра:

– строка **s [эс]**, в которую копируется символы;

– строка **s1 [эс один]**, из которой копируется;

– **n [эн]** – количество копируемых символов.

В первом примере показано: если копируется вся строка **s1 [эс один]**, то хвост строки **s [эс]** отбрасывается, строка **s [эс]**, как говорится, затирается.

Во втором примере показано: если копируется только часть строки **s1 [эс один]**, то заменяются первые символы строки **s [эс]**.

В результате работы этой функции строка **s [эс]** изменяется, а строка **s1 [эс один]** остается неизменной.

На слайде приведен пример, демонстрирующий работу этой функции. Массивы символов инициализированы при объявлении этих массивов.

Слайд 175

Функция конкатенации, или присоединения, объединяет две строки, присоединяет к строке **s [эс]** строку **s1 [эс один]**.

Внимательно рассмотрим структуру этой функции. Перед именем функции **strcat [стринг кэт]** стоит символ звездочка. Это означает, что функция возвращает результат в виде указателя. Тип функции **char [чар]** свидетельствует о том, что результатом работы этой функции будет указатель на строку символов.

В списке параметров этой функции в скобках указывается два параметра:

- строка **s [эс]**, к которой присоединяется;
- строка **s1 [эс один]**, которая присоединяется.

В результате работы этой функции строка **s [эс]** изменяется, а строка **s1 [эс один]** остается неизменной.

Рассмотрим пример на слайде. Так как при определении массива значения элементов массива определены, инициализированы, количество символов в массиве можно не указывать, оставив пустые квадратные скобки. Обращение к функции происходит в структуре функции вывода на печать. Если функция возвращает результат, то вызов функции допустим только в структуре какого-либо оператора. Функция возвращает указатель на начало строки **s [эс]**. Возврат из функции инициирует вывод на печать строки **s [эс]**.

Выведя на печать строку **s [эс]**, можно убедиться, что она изменилась.

Для использования этой функции необходимо подключить к программе заголовочный файл **string [стринг]**.

Слайд 176

Рассмотрим еще одну функцию конкатенации. В отличие от функции, представленной на предыдущем слайде, функция **strncat [стринг энд кэт]** присоединяет только часть строки **s1 [эс один]** к строке **s [эс]**.

Из описания структуры видно, что результатом работы этой функции является указатель. При обращении к этой функции указывается три параметра:

– строка **s [эс]**, к которой присоединяется;

– строка **s1 [эс один]**, которая присоединяется;

– **n [эн]** – количество символов, которые необходимо присоединить.

Пример на слайде демонстрирует работу этой функции. Обращение к функции происходит в операторе вывода на печать. Возвращается результат в виде указателя на начало строки **s [эс]**. Выведя на печать результат работы этой функции, можно убедиться, что только часть строки **s1 [эс один]** присоединилась к строке **s [эс]**. Строка **s1 [эс один]** остается неизменной. Можно присоединить всю строку, но при этом лучше воспользоваться функцией **strcat [стринг кэт]**, описанной на предыдущем слайде.

Внимательно проанализируйте приведенный на слайде пример, чтобы разобраться в особенностях применения этой функции.

Для использования этой функции необходимо подключить к программе заголовочный файл **string [стринг]**, который указан в первой строке таблицы.

Слайд 177

Представленная на слайде функция определяет длину первого сегмента в строке **s [эс]**, содержащего только символы из множества **s1 [эс один]**.

Из описания структуры понятно, что тип функции описан как **int [инт]** – целый. То есть результатом работы этой функции возвращаемый функцией

результат является целым числом. В скобках указываются два параметра, два символьных массива, две строки.

В приведенном примере видно, что функция возвращает целое число. Обращение к функции происходит в структуре функции вывода на печать. В нашем примере результат – число два, которое равно количеству подряд идущих символов строки **s1 [эс один]**, найденных в строке **s [эс]**. Несмотря на то, что некоторые символы далее в строке **s [эс]** встречаются, например, символ **k [кей]**, они в рассмотрение не попадают. Посимвольно сравниваются две строки, и как только символы не совпали, функция возвращает результат.

В результате работы этой функции строка **s [эс]** и строка **s1 [эс один]** остаются неизменными.

Если символы строки **s1 [эс один]** в строке **s [эс]** не найдены, результатом работы этой функции будет число ноль.

Для использования этой функции необходимо подключить к программе заголовочный файл **string [стринг]**.

Слайд 178

Функция, представленная на слайде, выполняет действия, противоположные действиям предыдущей функции. Функция определяет длину первого сегмента строки **s [эс]**, содержащего символы, не входящие в строку **s1 [эс один]**.

Из описания структуры видно – результатом работы этой функции является целое число, тип функции описан как **int [инт] –** целый. В скобках указываются два формальных параметра, два указателя типа **char [чар]**.

Обращение к функции происходит в операторе присваивания. Описана целая переменная **t [ти]**, инициализация значения этой переменной и есть

обращение к функции. В виде фактических параметров при обращении к функции указаны две строки.

В приведенном примере видно, что функция возвращает целое число, которое равно количеству символов строки **s [эс]**, не содержащих символов строки **s1 [эс один]**. Каждый символ строки **s1 [эс один]** ищется в строке **s [эс]**. Как только один из символов найден, функция возвращает результат. В приведенном на слайде примере это число три.

В результате работы этой функции строка **s [эс]** и строка **s1 [эс один]** остаются неизменными.

Если символы строки **s1 [эс один]** в строке **s [эс]** не найдены, результатом работы этой функции будет длина строки **s [эс]** без учета признака конца строки – символа **\0 [слеш ноль]**.

Для использования этой функции необходимо подключить к программе заголовочный файл **string [стринг]**.

Слайд 179

Функция **strstr [стринг стринг]** ищет в строке **s [эс]** подстроку **s1 [эс один]** и возвращает указатель на первое вхождение строки **s1 [эс один]** в строку **s [эс]**.

Из описания структуры видно – результатом работы этой функции является указатель на найденную строку. В скобках указываются два параметра, две строки:

- строка **s [эс]**, в которой ищется подстрока **s1 [эс один]**;
- строка **s1 [эс один]**, которая ищется.

В приведенном примере видно, что функция возвращает указатель на тот элемент в строке **s [эс]**, на тот символ, с которого начинается строка **s1 [эс один]**.

В результате работы этой функции строка **s [эс]** и строка **s1 [эс один]** остаются неизменными. Используя эту функцию совместно с **cout [си аут]**, на печать выводится часть строки **s [эс]** с положения указателя и до конца этой строки.

При решении различных задач можно использовать эту функцию поиска, оперируя указателем. Если совпадений не обнаружено, возвращается NULL.

Если необходимо продолжить поиск символов в исходной строке, достаточно сдвинуть внутренний указатель на следующий символ в строке и продолжить поиск. Этот принцип поиска мы рассмотрим далее на примере задач.

Для использования этой функции необходимо подключить к программе заголовочный файл **string [стринг]** с помощью препроцессорной команды.

Слайд 180

Функция **strchr [стринг чар]** ищет в строке символ с кодом **k [кей]** и возвращает указатель на элемент в строке **s [эс]** с кодом **k [кей]**.

Существует международная таблицы **ASCII [аски]** кодов. Таблица **ASCII [аски]** кодов – это таблица кодировки, в которой некоторым распространённым печатным и непечатным символам сопоставлены числовые коды. Таблица была разработана и стандартизована в США в 1963 году.

Таблица ASCII определяет коды для символов:

- десятичных цифр;
- латинского алфавита;

– национального алфавита;
– знаков препинания;
– управляющих символов.

Из описания структуры видно, что результатом работы этой функции является указатель. При вызове функции **strchr [стринг чар]** указывают два параметра:

– строка **s [эс]**, в которой ищется символ с кодом **k [кей]**;
– код искомого символа.

В приведенном примере видно, что функция возвращает указатель на первое вхождение символа с указанным кодом в строке. Где число 65 – код заглавной буквы латинского алфавита A.

В результате работы этой функции исходная строка остается неизменной.

При решении различных задач можно использовать эту функцию поиска, оперируя указателем.

Для использования этой функции необходимо подключить к программе заголовочный файл **string [стринг]**.

Слайд 181

Функция **strset [стринг сет]** заполняет строку **s [эс]** символами с кодом **k [кей]**. Функция возвращает указатель на начало строки **s [эс]**. Значение **k [кей]** указывается из таблицы **ASCII [аски]** кодов.

Из описания структуры видно – результатом работы этой функции является указатель на начало преобразованной строки символов.

Функции **strset [стринг сет]** имеет два параметра:

– строка **s [эс]**, которая заполняется символами с кодом **k [кей]**;
– код символа.

В приведенном примере видно, что функция заполняет строку **s [эс]** заглавной буквой латинского алфавита **B [бэ]** и возвращает указатель на начало строки **s [эс]**. Здесь 66 – код заглавной буквы латинского алфавита **B [бэ]**. При этом длина строки **s [эс]** остается неизменной.

В приведенном примере строка символов инициализирована при объявлении массива. Размер массива можно не указывать при объявлении массива. Он определяется длиной цепочки символов при инициализации. Поэтому рядом с именем массива можно указать пустые квадратные скобки.

Для использования этой функции необходимо подключить к программе заголовочный файл **string [стринг]**.

Слайд 182

Функция **strnset [стринг эн сет]** заменяет первые **n [эн]** символов в строке **s [эс]** символами с кодом **k [кей]**. Функция возвращает указатель на начало строки **s [эс]**. Значение числа **k [кей]** указывается из международной таблицы **ASCII [аски]** кодов.

Из описания структуры видно – результатом работы этой функции является указатель.

Функция **strnset [стринг эн сет]** имеет три параметра:

- строка **s [эс]**, которая заполняется символами с кодом **k [кей]**;
- код символа;
- количество символов.

В описании структуры этой функции первый формальный параметр определен как указатель символьного типа. А так как известно, что имя массива и есть указатель, при обращении к функции первым фактическим параметром является строка символов.

В приведенном примере видно, что функция заполняет три первых символа строки **s [эс]** заглавной буквой латинского алфавита A и возвращает указатель на начало строки. Число 65 – код заглавной буквы **A** латинского алфавита. В результате работы этой функции длина строки **s [эс]** остается неизменной.

Для использования этой функции необходимо подключить к программе заголовочный файл **string [стринг]**.

Слайд 183

Представленная на слайде функция преобразует буквы нижнего регистра в буквы верхнего регистра. Анализируются коды символов, согласно таблице **ASCII [аски]** кодов.

Из описания структуры функции видно, что результатом работы этой функции является преобразованная строка. Указатель устанавливается на начало преобразованной строки. В конечном итоге функция возвращает указатель на начало преобразованной строки.

Функция имеет один параметр – строку символов, которую необходимо преобразовать.

В приведенном примере малые буквы латинского алфавита преобразованы в заглавные. Если в цепочке символов встретятся заглавные буквы, то они остаются неизменными.

Вызов этой функции реализован в структуре функции вывода на печать, соответственно, преобразованная строка символов распечатается на экране.

Длина строки **s [эс]** остается неизменной.

Для использования этой функции необходимо подключить к программе заголовочный файл **string [стринг]**.

Слайд 184

Тема 7. Массивы символов. Строки (часть 2)

Представленная на слайде функция преобразует буквы верхнего регистра в буквы нижнего регистра.

Из описания структуры функции видно, что результатом работы этой функции является указатель на начало преобразованной строки.

Функция имеет один параметр – строку символов, которую необходимо преобразовать.

В приведенном примере заглавные буквы латинского алфавита преобразованы в малые.

Функция устанавливает внутренний указатель на начало преобразованной строки. Вызов этой функции реализован в структуре функции вывода, соответственно, преобразованная строка символов распечатается на экране.

Длина строки **s [эс]** остается неизменной.

Для использования этой функции необходимо подключить к программе заголовочный файл **string [стринг]**.

Слайд 185

Представленную на слайде функцию мы уже применяли ранее при решении некоторых задач. Рассмотрим работу этой функции подробно.

Из описания структуры видно – результатом работы этой функции является целое число, тип функции описан как **int [инт]**. Функция посимвольно сравнивает коды двух строк – строку символов **s1 [эс один]** и строку **s2 [эс два]** и возвращает значение ноль, если коды совпали. В таблице **ASCII [аски]** кодов коды заглавных и прописных букв отличаются. Если

одно и то же слово написано в одной строке заглавными, а в другой строке прописными, сравнение не пройдет.

Если сравнение строк не прошло, функция возвращает число, которое можно использовать в программе для дальнейшего анализа. На слайде приведены возможные значения, возвращаемые этой функцией.

Функция имеет два параметра – две строки, которые сравниваются. На слайде показано, какие значения может возвращать эта функция.

В результате работы этой функции строка **s1 [эс один]** и строка **s2 [эс два]** остаются неизменными.

Ниже мы рассмотрим примеры с использованием этой функции.

Для использования этой функции необходимо подключить к программе заголовочный файл **string [стринг]**.

Слайд 186

На слайде представлена функция инвертирования строки символов.

Из описания структуры функции видно, что перед именем функции стоит символ звездочка. Следовательно, результатом работы этой функции является указатель на начало инверсированной строки символов.

Функция имеет один параметр – строку символов.

В приведенном примере описан массив символов на сто элементов – символов. С помощью функции **gets [гет стринг]** вводится строка символов длиною не больше ста элементов. По завершении ввода символов в конце строки автоматически добавится признак конца строки – символ **\0 [слеш ноль]**.

На слайде приведен пример, где функция инвертирования указана как аргумент функции **puts [пут стринг**] вывода строки символов на экран.

Вывод преобразованной строки **s [эс]** осуществляется до признака конца строки.

Для использования этой функции необходимо подключить к программе заголовочный файл **string [стринг]**.

Слайд 187

Функция **strlen [стринг лен]** определяет длину строки символов. Без этой функции практически не обходится ни одна задача обработки строк символов. Можно использовать символьный массив, но намного удобнее работать со строкой символов и обрабатывать всю строку целиком.

Из описания структуры видно – результатом работы этой функции является целое число, определяющее количество символов в строке **s [эс]**.

Функция имеет один параметр – строка символов. Возвращаемый функцией результат – целое число, длина строки символов.

В приведенном примере определена строка символов, заданная в двойных кавычках. При такой инициализации массива в конце строки автоматически вставляется признак конца строки – символ **\0 [слеш ноль]**. При подсчете длины строки этот символ не учитывается. Обращение к функции происходит в структуре функции вывода на печать.

Исходная строка символов остается неизменной.

Для использования этой функции необходимо подключить к программе заголовочный файл **string [стринг]**. Имя файла указано в верхней строке таблицы.

Слайд 188

Приведем примеры задач практического применения рассмотренных ранее функций обработки строк символов.

На данном слайде продемонстрирован фрагмент, позволяющий подсчитать, сколько раз в тексте встретился восклицательный знак.

Описан одномерный массив символов.

Для ввода строки символов используется функция ввода **gets [гет стринг]**, ввод символов завершается нажатием клавиши **enter [энтер]**. В конце строки автоматически вставляется символ **\0 [слеш ноль]** – признак конца строки**.**

В цикле подсчитывается, сколько раз встретился искомый символ.

В структуре оператора цикла указано условие прекращения цикла – функция **strlen [стринг лен]**. Эта функция подсчитывает длину строки без признака конца строки. В приведенном примере строка рассматривается как обычный одномерный массив, анализируется каждый элемент массива.

Слайд 189

Рассмотрим задачу. Нужно подсчитать количество предложений в тексте.

Для использования функций обработки строк символов необходимо подключить заголовочные файлы:

– **iostream [ай о стрим**] – для вывода информации на экран;

– **stdio [эс ти дио]** – для функции ввода строки символов **gets [гет стринг]**;

– **string [стринг]** – для функции определения длины строки символов **strlen [стринг лен]**.

Предложение может заканчиваться одним из следующих знаков: точкой, вопросительным знаком, восклицательным знаком. Исходя из этого, мы и будем решать задачу. Строка символов вводится с клавиатуры. Ввод

завершается нажатием клавиши **Enter [энтер]**. Введенный текст рассматривается как одномерный массив символов.

Подсчет предложений в тексте производится в цикле, количество циклов определяется длиной введенного текста. Длина текста вычисляется функцией **strlen [стринг лен]**.

На слайде продемонстрирован фрагмент программы.

Слайд 190

На слайде представлена формулировка задачи. Введены обозначения используемых переменных.

Из исходного текса будем выделять количество символов, равных длине искомого слова. Под искомым словом понимается любая цепочка символов. Выделенные из исходного текста символы попадают в переменную – буфер слов и сравниваются с искомым словом.

Таким образом, перемещаясь по тексту сначала с первого символа, затем со второго символа и так далее, подсчитаем, сколько раз встретилось искомое слово. Для сравнения будем использовать функцию посимвольного сравнения выделенных из текста символов с искомым словом.

Последний раз необходимо выделить символы исходного текста так, чтобы точно уложиться в этот текст, не выходя за его границы.

Слайд 191

На слайде представлен первый фрагмент программы решения задачи.

Подключены заголовочные файлы, необходимые для использования функций ввода-вывода строки символов, функции определения длины строки, функции сравнения строк.

Описаны массивы символов для исходного текста, искомого слова и буфера слов. Размер буфера равен длине искомого слова.

С клавиатуры вводится исходный текст и искомое слово. Используется функция ввода строки символов **gets [гет стринг]**. В конце строки автоматически введется признак конца строки – константа **\0 [слеш ноль]**. Это позволит в следующем фрагменте программы использовать функции обработки строк.

Слайд 192

Введенный с клавиатуры текст будем рассматривать как строку символов. Введена переменная-указатель **p [пи]**, которая устанавливается в начало исходного текста. Обратите внимание на то, как описана и определена эта переменная. Указателю присваивается адрес начала строки символов.

Перемещаясь по тексу от символа к символу, функция **strncpy [стринг эн копи]** выделяет из искомого текста в буфер переменную **s [эс]** – количество символов, равных длине искомого слова.

Функция **strcmp [стринг си эм пи]** сравнивает коды символов буфера с искомым словом. Если прошло сравнение, эта функция возвращает значение, равное нолю. Переменная **k [кей]** подсчитывает, сколько раз встретилось искомое слово в тексте.

На слайде введены обозначения:

– 1 – выделение символов из исходного текста в буфер **s [эс]** начиная с позиции указателя **p [пи]**;

– 2 – сравнение выделенных символов с искомым словом.

Параметрами функции **strncpy [стринг эн копи]** являются:

– **s [эс]** – буфер слов, строка, в которую копируются выделенные символы;

– **p [пи]** – указатель, позиция, с которой из исходного текста выделяются символы;

– **strlen(word) [стринг лен ворд]** – количество выделяемых из исходного текста символов, равное длине искомого слова.

Ранее мы рассматривали принцип работы этой функции.

Обратите внимание, что после каждого цикла значение указателя **p [пи]** увеличивается на единицу. Переходим к следующему символу исходного текста, и процесс поиска продолжается.

Нет необходимости очищать буфер слов – строку **s [эс]**, так как количество символов, которые копируются из исходного текста, все время одинаковое, равное длине искомого слова. На каждом шаге новые символы как бы затирают старые символы.

Слайд 193

Рассмотрим задачу удаления части текста.

Введены обозначения: исходный текст, часть текста, которую необходимо удалить, преобразованный текст. Часть текста, которую надо удалить, будем называть просто словом.

Для решения этой задачи поиска слова в исходном тексте будем использовать функцию поиска **strstr [стринг стринг]**. Ранее мы рассматривали функцию поиска. Результатом работы этой функции является указатель. Начиная с этого указателя, будем удалять часть исходного текста.

Функция конкатенации, или присоединения, сформирует новый текст из исходного. К части текста до удаляемого слова добавляется часть текста после слова.

Управляя указателем, как результатом работы этих функций, сформируем новый текст.

На следующих слайдах представлена программа решения этой задачи.

Слайд 194

Подключенные заголовочные файлы необходимы для применения перечисленных выше функций.

Исходный текст и удаляемое слово, или часть текста, описаны как массивы символьного типа. Конкретные размеры будут определены во время ввода с клавиатуры исходного текста и удаляемого слова. Ввод осуществляется с помощью функции **gets [гет стринг]**.

Ввод строки символов завершается нажатием клавиши **Enter [энтер]**. При этом в конце строки, определяющей исходный текст, и в конце строки, определяющей удаляемое слово, автоматически формируется признак конца строки.

Обратите внимание, что текст, который будет сформирован из исходного текста, то есть **newText [нью текст]**, описан как символьный массив без указания размера. Пустые квадратные скобки рядом с именем массива вполне допустимы. Размер этого массива заранее не известен. Однако в пустой текст введен признак конца строки – символ **\0 [слеш ноль]**. Так как новый текст будет формироваться с помощью функции конкатенации, присутствие признака конца строки необходимо. При использовании функции конкатенации признак конца строки автоматически не добавляется.

Слайд 195

Функция **strstr [стринг стринг]** ищет слово, которое необходимо удалить из текста. Результат работы этой функции – указатель на начало найденного слова. Под словом понимают любую цепочку символов.

Введена переменная **p [пи]**, указатель, для поиска слова в исходном тексте. Анализируется значение указателя, и если слово не найдено в тексте, появится соответствующее сообщение.

Если слово найдено, функция конкатенации **strncat [стринг эн кэт]** присоединит часть исходного текста до удаляемого слова к новому тексту. Признак конца слова, который внесен в новый массив в начале программы, автоматически отодвигается.

Указатель устанавливается в начало найденного слова в исходном тексте. Далее указатель **p [пи]** отодвигается в тексте на длину удаляемого слова.

Еще раз, используя функцию конкатенации, присоединяем к новому тексту остаток исходного текста после слова, если оно не последнее. Признак конца автоматически отодвигается.

Функция **puts [пут стринг]** выведет на экран новый текст до признака конца.

Слайд 196

Рассмотрим определение симметричности слова. Симметричными считаются слова, которые одинаково читаются слева направо и справа налево, так называемые палиндромы.

На слайде сформулирована задача и показаны введенные обозначения.

Это исходный текст и буфер слов, в который выделяется каждое слово текста и анализируется на симметричность. Они определены как массивы символов. Признаком конца слова считается символ пробел.

Конечно, на самом деле признаками конца слова являются и запятая, и двоеточие, и точка, если текст состоит из нескольких предложений, и другие знаки. Чтобы не загромождать программу, в этом примере признаком конца слова считается символ пробел. Добавить дополнительные проверки на другие символы не составит сложности.

Каждое отдельное слово будет выделено в буфер – переменную **word [ворд]** и проверено на симметричность.

Переменная **f [эф]** введена как признак симметричности.

Слайд 197

На слайде представлен первый фрагмент программы. Подключены заголовочные файлы для использования функций обработки символьных массивов, или строк.

Описаны исходный текст и буфер слов как одномерные массивы символов. Каждое слово текста попадает в переменную **word [ворд]**.

Для ввода исходного текста используется функция **gets [гет стринг]**. Напомним, что ввод цепочки символов завершается нажатием клавиши **Enter [энтер]**. Если цепочка символов, то есть исходный текст, достаточно длинная, курсор сам перейдет на начало следующей строки, и продолжается ввод.

Переменная **p [пи]** определена как начало каждого слова, то есть позиция начала каждого слова в исходном тексте. Фактически это индекс одномерного массива символов. Переменная описана как целая, так как известно, что индексы массивов должны иметь целый тип.

Известно, что в любом массиве есть нулевой элемент. Поэтому этой переменной присвоено значение ноль – начало первого слова в исходном тексте.

Слайд 198

Для поиска симметричных слов используется структура вложенных циклов: внешнего – цикла с предусловием, двух внутренних – циклов с параметром.

Как говорилось выше, переменная **p [пи]** – позиция начала каждого слова. Внешний цикл продолжается до тех пор, пока не достигли конца исходного текста.

Функция **strlen [стринг лен]** определяет длину исходного текста.

В первом внутреннем цикле, реализованном оператором **for [фо]**, каждый символ исходного текста переносится посимвольно в буфер слова от пробела до пробела.

Переменная **i [ай]** используется как индекс исходного текста, значение которого на каждом шаге итерации увеличивается на единицу.

Переменная **j [джей]** используется как индекс буфера слов, значение которого обнуляется перед каждым переносом очередного слова в буфер слов.

Первый внутренний цикл завершается, как только очередное слово до пробела перенесено в буфер.

Индекс **j [джей]** определен для буфера слов. Индекс **i [ай]** меняется в исходном тексте.

В переменную, определенную как буфер слов, добавляется признак конца строки – символ **\0 [слеш ноль]**. Формируется признак конца строки. В операторе **for [фо]** значение переменной **j [джей]** после завершения цикла на единицу больше, чем длина очередного выделенного слова. В предыдущих темах мы подробно рассматривали принцип работы оператора **for [фо]**. Именно в эту позицию заносится признак конца строки.

После каждого слова переменная **p [пи]** переносится на начало следующего слова в исходном тексте.

Слайд 199

На слайде представлен следующий фрагмент программы. Тип переменной **f [эф]** определен как логический. Обратите внимание, что перед

каждым словом значению этой переменной присваивается значение **true [тру]** – истина. Предполагается, что слово симметрично.

Анализ каждого слова на симметричность выполняется во втором внутреннем цикле. Слово мысленно делится пополам и посимвольно сравниваются символы с левой и с правой сторон очередного слова. Функция **strlen [стринг лэн]** определяет длину очередного слава. Если символы не совпали, выставляется признак несимметричности, то есть переменной **f [эф]** присваивается значение **false [фолс]** – ложь. Количество проверок равно половине длины слова.

По окончании второго внутреннего цикла, в зависимости от значения переменной **f [эф]** выводятся на экран симметричные слова исходного цикла. Фигурная скобка закрывает внешний цикл.

Мы рассмотрели примеры, позволяющие осуществлять поиск в тексте, удалять часть текста, проверять текст на симметричность, чтобы продемонстрировать работу функций обработки массивов.

На этом мы завершаем изучение темы «Символьные массивы, строки».

Слайд 200

Тема 8. Структуры, объединения, перечисления

Кроме базовых типов данных, рассмотренных в предыдущих темах, язык **C++ [си плюс плюс]** предоставляет простые и структурированные типы данных.

«Используя базовые, простые и структурированные типы данных, можно создавать производные типы данных, так называемые пользовательские типы данных.

Напомним, что к базовым типам относятся целочисленный тип, вещественный тип, логический тип, символьный тип, строковый тип.

К структурированным типам относятся:

– массивы;
– структуры;
– файлы;
– объектный тип» [5].

Перечисления – средство создания типа данных посредством задания ограниченного множества значений. Значения данных перечисляемого типа указываются идентификаторами. Перечислением называется тип данных, который включает множество именованных целочисленных констант. Именованные константы, принадлежащие перечислению, называются перечислимыми константами.

Для объявления перечислений используется ключевое слово **enum [энам].**

Имя перечисления – это уникальный идентификатор, определяющий тип перечисления, это пользовательский тип данных.

Имена констант являются элементами перечисления.

Объявление типа перечисления и переменной, которая имеет этот тип, может быть объединено в одну инструкцию. Пример объявления перечисления приведен на слайде.

color [колор] – имя типа перечисления, пользовательский тип.

r [ар], g [джи], b [би] – сами перечислимые константы, элементы перечисления.

Слайд 201

«Перечислимый тип описывает множество, состоящее из элементов-констант, иногда называемых нумераторами или именованными константами.

Значение каждого нумератора определяется как значение типа **int [инт]**, то есть целый. По умолчанию первый нумератор определяется значением ноль, второй – значением единица и так далее» [4].

Для инициализации значений нумератора не с нуля, а с другого целочисленного значения следует присвоить это значение первому элементу списка значений перечислимого типа. Для перечислимого типа существует понятие диапазона значений, определяемого как диапазон целочисленных значений. Это первый пример на слайде.

При объявлении типа перечисления его значения могут инициализироваться произвольными целочисленными константами или константным выражением, как это показано в первом примере на слайде.

Если инициализация отсутствует, то перечислимым константам присваиваются последовательные значения, начиная от нуля, как показано во втором примере.

Если инициализированы не все элементы перечисления, то значения элементов, которые не определены, будут распределены соответственно, как показано в третьем примере

Переменная типа перечисление объявляется, как показано в четвертом примере в таблице, например, объявление переменной **d [ди]**.

Объявление типа перечисления и переменной, которая имеет этот тип, может быть объединено в одну инструкцию, как показано в пятом примере.

Переменным перечислимого типа можно присваивать только именованные значения перечислимых констант. Этот пример приведен в шестой строке в таблице. При необходимости можно явно задавать значения идентификатора, тогда очередные элементы списка получат последующие возрастающие значения.

В строке семь – ошибочное присвоение перечислимой константе. Целочисленным переменным можно присваивать только значения перечислимых констант.

Слайд 202

«В программировании есть много случаев, когда может понадобиться больше одной переменной для представления определенного объекта.

К счастью, язык **C++ [си плюс плюс]** позволяет программистам создавать свои собственные пользовательские типы данных – типы, которые группируют несколько отдельных переменных вместе. Одним из простейших пользовательских типов данных является структура. Структура позволяет сгруппировать переменные разных типов в единое целое.

Переменные типа структура подчиняются тем же правилам, что и обычные переменные. Следовательно, если вы хотите сделать переменную структуры доступной в нескольких файлах, то вы можете использовать ключевое слово **extern [экстерн]**» [4].

Тип структура описывает упорядоченный набор данных, которые называются полями или элементами структуры. Каждое поле структуры имеет имя и тип, который должен отличаться от типов **void [войд]**. При этом следует учитывать, что структура может включать только такие поля, длина которых известна компилятору в момент определения структуры. Элементами структуры могут быть массивы и другие структуры.

При обработке большого объема прибегают к объединению информации в массивы. Недостаток массивов – это возможность объединения в массивы только однотипной информации, одинакового типа.

При создании базы данных возникает необходимость в объединении информации разного типа.

Итак, структура – это объединение в единое целое множества поименованных элементов разного типа. Описание структуры выполняется с помощью зарезервированного слова **struct [структ]**. Это пользовательский тип данных, который строится на базе стандартных типов данных, допустимых на языке **C++ [си плюс плюс]**. Имя структуры – уникальный идентификатор. Имена структур принято писать с заглавной буквы, чтобы отличать их от имен переменных.

Слайд 203

Каждая структура включает в себя один или несколько объектов, называемых элементами структуры. Каждый элемент имеет тип и имя элемента. Описание каждого элемента заканчивается точкой с запятой. Элементы структуры заключаются в фигурные скобки, после которых также ставится точка с запятой. Отметим, что структуры даже одного типа нельзя сравнивать между собой.

Нельзя забывать ставить точку с запятой в конце объявления структуры. Это приведет к ошибке компиляции в следующей строке кода.

На слайде приведен пример структуры – базы данных книголюба.

Имя структуры **buk [бук] –** пользовательский тип данных. Описанная структура состоит из четырех элементов:

– фамилия автора книги;

– название книги;

– год издания;

– количество страниц.

Определены имя и тип каждого поля структуры.

На слайде с помощью пользовательского типа описаны:

– структура buk [бук];

– массив структур a [эй];

– указатель на структуру pbuk [пи бук].

С именем структуры не связан конкретный объект, и с его помощью нельзя обращаться к элементам структуры. На слайде приведены примеры обращения к элементам структуры.

Первый пример – обращение к первому символу в фамилии автора.

Второй пример – год издания некоторой книги.

«Бóльшим преимуществом использования структур, нежели отдельных переменных, является возможность передать всю структуру в функцию, которая должна работать с её членами. Это позволило нам не передавать каждую переменную по отдельности. Функция также может возвращать структуру. Это один из тех немногих случаев, когда функция может возвращать несколько переменных» [4].

Слайд 204

«Поскольку объявление структуры не провоцирует выделение памяти, то использовать предварительное объявление для нее вы не сможете. Если вы хотите использовать объявление структуры в нескольких файлах, чтобы

иметь возможность создавать переменные этой структуры в нескольких файлах, поместите объявление структуры в заголовочный файл. Подключайте этот файл везде, где необходимо использовать структуру» [3].

Как и в случае с обычными переменными, элементы структуры не инициализируются автоматически и обычно содержат мусор. Инициализировать их нужно вручную.

Инициализация структур путем присваивания значений каждому члену по порядку – занятие довольно громоздкое, особенно если этих членов много. В языке **C++ [си плюс плюс]** есть более быстрый способ инициализации структур – с помощью списка инициализаторов. Он позволяет инициализировать некоторые или все члены структуры во время объявления переменной типа **struct [структ]**.

Если в списке инициализаторов не будет одного или нескольких элементов, то им присвоятся значения по умолчанию, обычно это ноль» [4].

Если структура определяется один раз в программе, то при описании структуры можно опустить имя структуры, как это показано на слайде. После определения структуры описаны переменные типа структура.

На слайде приведен пример инициализации элементов структуры.

Так как имя структурного типа обладает всеми правами типов переменных, то разрешено определять указатели на структуры.

Указатель может быть инициализирован. Значением указателя на структуру является адрес, а именно – номер байта, начиная с которого структура размещается в памяти.

Указатель задает размер структуры и тем самым определяет, на какую величину, то есть на сколько байт, перемещается указатель на структуру, если к нему добавить или отнять единицу.

На слайде описан указатель на структуру с именем **pbuk [пи бук]**.

Слайд 205

«Объединение – это группирование переменных, которые разделяют одну и ту же область памяти. В зависимости от интерпретации осуществляется обращение к той или другой переменной объединения. Все переменные, что включены в объединение, начинаются с одной границы.

Объединение позволяет представить в компактном виде данные, которые могут изменяться. Одни и те же данные могут быть представлены разными способами с помощью объединений.

Тип переменной, входящий в объединение, может быть:

- базовым типом;
- типом структура;
- типом объединение;
- типом класс.

Объединение – это структура данных, члены которой расположены по одному и тому же адресу. Поэтому размер объединения равен размеру его наибольшего члена. В любой момент времени объединение хранит значение только одного из членов» [4].

Итак, объединения – это объект, позволяющий нескольким переменным различных типов занимать один участок памяти. Объявление объединения похоже на объявление структуры. Тип объединение описывает набор данных, которые называются элементами или членами объединения. Объединение позволяет хранить в одной и той же области памяти значения различных типов, которые соответствуют типам элементов объединения.

Как и элементы структуры, элементы объединения могут иметь любой тип за исключением типов **void [войд]** и функция.

Для объявления объединения используется ключевое слово **union [юнион]**.

Пример объявления объединения приведен на слайде:

– name [нейм] – имя объединения, это пользовательский тип, уникальный идентификатор;

– n [эн], f [эф] – элементы объединения.

Каждый элемент объединения имеет имя и тип.

Описана переменная **d [ди]**, имеющая тип – объединение.

Слайд 206

«Использование объединения обозначает, что имеется намерение задействовать только одну ячейку памяти компьютера, в которую будут подставляться значения переменных, объединённых в одну группу и, вероятно, имеющих разные типы. Типы в **C++ [си плюс плюс] –** не что иное, как подсказка компьютеру, сколько байтов памяти нужно отвести под переменную. Самый широкий тип, описанный внутри объединения, считается за подсказку, сколько памяти нужно выделить объединению. Объединения подобны структурам и пишутся так же, как и структуры.

Длина объединения – это размер памяти в байтах, которая выделяется для одной переменной этого типа объединения» [4].

Объявление типа объединения и переменной, которая имеет этот тип, могут быть объединены в одну инструкцию, как показано в первом примере.

При объявлении переменной типа объединение её можно инициализировать значением, которое должно иметь тип первого члена объединения. Во втором примере описана и инициализирована переменная **d [ди]**.

Третий пример – ошибочный, так как первый элемент объединения имеет целый тип.

Как и в случае со структурами, для доступа к элементу объединения используется точка, как показано в четвертом примере.

Объединения одного типа можно присваивать друг другу. В этом случае оператор присваивания выполняет почленное копирование объединений.

Так же, как и структуры, объединения нельзя сравнивать.

Перейдем к задачам, позволяющим на примерах рассмотреть изложенный выше материал.

Слайд 207

Элементы структуры называются полями структуры и могут иметь любой тип, кроме типа этой же структуры, но могут быть указателями на него. Если отсутствует имя типа, должен быть указан список описателей переменных, указателей или массивов. В этом случае описание структуры служит определением элементов этого списка. Еще раз вспомним, что массив – это поименованный набор однотипной информации.

Создание структур позволяет хранить в массиве информацию разного типа. На слайде приведена формулировка задачи, в которой создается

массив, элементы которого – структура. Создается массив структур. Каждый элемент массива хранит информацию об одном наименовании товара.

Структура объединяет информацию разного типа: символьного типа, целого типа и вещественного типа.

Составим программу, которая позволит:

– создать массив структур, содержащий информацию о товарах некоторого склада;

– по введенному с клавиатуры наименованию вывести информацию о соответствующем товаре или вывести сообщение об отсутствии данного товара на складе;

– подсчитать общую стоимость товаров на складе.

Необходимо ввести переменную, определяющую количество наименований товаров, и переменную для подсчета общей стоимости товаров.

Слайд 208

На слайде представлен первый фрагмент программы.

Подключены заголовочные файлы:

– **iostream [ай о стрим]** – для организации ввода-вывода информации: заполнения базы данных, массива структур и вывода результатов вычислений;

– **string [стринг]** – для сравнения наименований товаров;

– **conio [конио]** – для использования функции **getch [гетч]**.

Определена структура – пользовательский тип с именем **Tovar [товар]**.

Описан массив структур. Размер каждого элемента массива определен описанной структурой:

– переменная **name [нейм]** занимает 30 байт;

– переменная **kol [кол] – 4 байта**;
– переменная **price [прайс] – 4 байта**.

Описанный массив структур состоит из тридцати элементов, где размер каждого элемента массива в байтах определяется размером структуры.

Еще раз напомним, что количество байт, отводимых под каждый тип информации, например целый тип, зависит от конкретной архитектуры компьютера.

Слайд 209

Для переменных одного и того же структурного типа определена операция присваивания, при этом происходит поэлементное копирование. Структуру можно передавать в функцию и возвращать в качестве значения функции. Другие операции со структурами могут быть определены пользователем. Размер структуры не обязательно равен сумме размеров ее элементов, поскольку они могут быть выровнены по границам слова.

Количество наименований товаров, количество элементов в массиве структур определяет целая переменная **n [эн]**, значение которой вводится с клавиатуры.

Вещественная переменная **s [эс]** введена для подсчета общей стоимости всех товаров.

Переменная **name [нейм] –** наименование искомого товара – описана как символьный массив. Размер этой переменной, этого массива должен совпадать с размером элемента структуры, отведенного под хранение наименований товаров.

В нашем примере имя этой переменной совпадает с именем первого элемента структуры. Имена могут и не совпадать, а вот размеры – должны быть одинаковыми при описании, так как сравнение предполагается

побайтно. Сравниваться будут коды символов в наименованиях товаров из массива структур с кодами в наименовании искомого товара.

Совпадение имен не внесет разночтений, так как при обращении к элементам структуры присутствует имя массива и через точку указывается имя соответствующего элемента поля.

Рядом с именем массива в квадратных скобках указывается индекс. Это фактически номер одного наименования товара в общем массиве структур.

В цикле вводится информация и создается массив структур. На каждой итерации цикла вводится информация об одном наименовании товара.

Слайд 210

На слайде представлен последний фрагмент программы.

Переменная **k [кей]** введена как признак наличия в массиве, то есть базе данных, искомого товара. Первоначально этой переменной присваивается значение минус единица. Известно, что массив не может иметь отрицательного значения индекса. Если товар в массиве найден, то переменной **k [кей]** присваивается номер элемента с искомым товаром.

В операторе цикла определена переменная **i [ай]** как параметр цикла.

Сравнивается наименование товара каждого элемента массива с наименованием искомого товара. Функция сравнения символьных массивов вырабатывает значение, равное нулю, если прошло сравнение. Особенности этой функции мы рассматривали в предыдущей теме. Обращение к элементу структуры осуществляется через имя массива структур – **array [аррей]** и имя конкретного поля структуры – **name [нейм]**. Индекс **i [ай]** определяет номер элемента в массиве структур.

Если наименование очередного товара совпало с наименованием искомого товара, переменной **k [кей]** присваивается номер этого элемента массива. Одновременно в этом же цикле подсчитывается общая стоимость

товаров, информация о которых хранится в массиве структур. Для подсчета общей стоимости товаров определена переменная **s [эс]**.

По завершении цикла анализируется значение переменной **k [кей]** и выдается информация о данном товаре по сохраненному номеру. Если значение этой переменной равно минус единице, значению, присвоенному в начале программы, выдается сообщение об отсутствии такого товара.

Слайд 211

Рассмотрим еще один пример обработки массива структур. На слайде определена структура массива, содержащая информацию о студентах некоторой группы.

Необходимо отсортировать массив структур. При сортировке необходимо выбрать ключ, по которому производится сортировка. Таким ключом в данной задаче является фамилия студентов. Будут сравниваться коды первой буквы в фамилиях. Напомним, что коды символов указаны в международной таблице **ASCII [аски]** кодов. Список нужно отсортировать в алфавитном порядке по фамилиям студентов.

Каждая структура состоит из трех элементов. Первое поле структуры – массив символов, отведенный под хранение фамилий студентов.

Второе поле структуры содержит информацию целого типа – дата рождения.

Третьим элементом структуры является массив, содержащий результаты сдачи экзаменов студентами по трем дисциплинам. Размер массива зависит от количества сданных студентами экзаменов. Размер всей структуры определяется входящими в нее элементами, количеством студентов в группе.

Слайд 212

На слайде представлен первый фрагмент программы. Подключены заголовочные файлы.

Обычные массивы хранят однотипную информацию. Если необходимо хранить и обрабатывать информацию разного типа, прибегают к построению структур.

На слайде описаны следующие элементы структуры:

– фамилия студента, информация символьного типа. Под фамилию выделено 20 символов;

– дата рождения, целочисленный тип;

– оценки, полученные на экзаменах, массив целых чисел.

Элементами структуры могут быть объекты различных типов, массивы и даже структуры. Как известно, структура – это пользовательский тип данных, базирующийся на стандартных типах данных языка **C++ [си плюс плюс]**.

Описан пользовательский тип данных – структура с именем **Student [студент]**. Информация будет храниться в массиве структур.

Слайд 213

Рассмотрим следующий фрагмент программы. В данном фрагменте описан массив структур. Количество элементов в массиве зависит от количества студентов в группе.

В программе введены следующие обозначения:

– **n [эн]** – количество студентов в группе;

– **buf [буф]** – переменная для обмена местами соседних элементов при сортировке, имеет тип структура. Обмениваться будут не отдельные поля структур, а целиком вся структура;

– **st_array [стринг эррей]** – массив структур.

С клавиатуры вводится количество студентов, то есть значение переменной **n [эн]**. В цикле продемонстрирован ввод информации о студентах группы. На каждой итерации цикла вводится информация об одном студенте: фамилия, дата рождения и оценки по трем экзаменам. Для ввода оценок каждого студента организован еще один внутренний цикл.

Обратите внимание, что при обращении к отдельным элементам структуры указывается сначала имя структуры и через точку имя соответствующего элемента структуры.

Слайд 214

Сортировка массива структур выполняется во вложенных циклах. Введены обозначения:

– 1 – внутренний цикл;

– 2 – внешний цикл.

Внутренний цикл позволяет один раз пройтись по массиву структур, сравнивая соседние элементы. При обращении к массиву структур используют индекс массива, переменную **i [ай]**. Это номер студента в списке группы. Индекс ноль, указанный в переменной **name [нейм]**, означает, что сравниваются коды первых символов в фамилиях студентов. Напоминаем о наличии нулевого элемента в массиве.

Обмен соседних элементов массива выполняется через переменную **buf [буф]**, которая имеет размер структуры. Вся информация об одном студенте – это один элемент массива структур.

Обратите внимание на закон изменения параметра внутреннего цикла, переменную **i [ай]**. На слайде выделено и подчеркнуто условие прекращения цикла. Последний раз в сравнении участвует предпоследний и последний элементы массива структур.

Внешний цикл продолжается до полной сортировки массива структур по элементу фамилия студента.

Функция **swap [свап]**, которую мы часто использовали для обмена переменных, не всегда адекватно работает со структурами. Поэтому используем переменную буфер для обмена соседних элементов структуры.

Слайд 215

Последний фрагмент программы демонстрирует вывод на экран массива структур, отсортированного в алфавитном порядке по фамилиям студентов. На экран выводится соответствующее сообщение.

В программе используется структура вложенных циклов.

Во внешнем цикле выводится информация об одном студенте. Вывод экзаменационных оценок осуществляется во внутреннем цикле.

Для вывода оценок используется два параметра:

– параметр внешнего цикла **i [ай]** определяет номер студента в списке группы, в отсортированном массиве;

– параметр внутреннего цикла **j [джей]** перебирает все оценки, полученные данным студентом во время экзаменационной сессии.

Вывод организован так, что фамилия и дата рождения выводятся на одной строке, а затем в столбик выводятся экзаменационные оценки. Управлять выводом можно с помощью управляющих символов перевода курсора на новую строку и управляющим символом табуляция.

Слайд 216

Мы уже рассмотрели такие понятия, как переменные-указатели, динамические массивы, рекурсивные функции, структуры, массивы структур.

Ознакомившись с перечисленными темами, рассмотрим динамические структуры данных, которые базируются на перечисленных понятиях.

«Структуры одного типа можно объединять не только в массивы. Их можно связывать между собой, создавая так называемые динамические структуры данных. Связь между отдельными структурами может быть организована по-разному, и именно поэтому среди динамических данных выделяют списки, стеки, очереди, деревья, графы. Для динамических данных память выделяется и освобождается в процессе выполнения программы, а не в момент ее запуска.

Список – абстрактный тип данных, реализующий упорядоченный набор значений. Списки отличаются от массивов тем, что доступ к их элементам осуществляется последовательно, в то время как массивы – структура данных произвольного доступа.

Список – это структура данных, представляющая собой конечное множество упорядоченных элементов, связанных друг с другом посредством указателей. Каждый элемент структуры содержит поле с какой-либо информацией, а также указатель на следующий элемент. В отличие от массива, к элементам списка нет произвольного доступа.

Так, если в программе объявлен массив из 100 элементов, при запуске программы резервируется память для всех ста элементов, даже если в процессе работы программы всего будут использованы первые 10 элементов массива. С другой стороны, при использовании в программе динамических типов память под них заранее не выделяется. Лишь когда поступают новые данные, вызывается специальная функция, которая выделяет память, куда эти данные записываются» [1].

Схема связей динамических структур представлена на этом слайде.

Слайд 217

«Динамические структуры данных в процессе существования в памяти могут изменять не только число составляющих их элементов, но и характер связей между элементами. При этом не учитывается изменение содержимого самих элементов данных. Такая особенность динамических структур, как непостоянство их размера и характера отношений между элементами, приводит к тому, что на этапе создания машинного кода программа-компилятор не может выделить для всей структуры в целом участок памяти фиксированного размера, а также не может сопоставить с отдельными компонентами структуры конкретные адреса. Для решения проблемы адресации динамических структур данных используется метод, называемый динамическим распределением памяти, то есть память под отдельные элементы выделяется в момент, когда они начинают существовать в процессе выполнения программы, а не во время компиляции. Компилятор в этом случае выделяет фиксированный объем памяти для хранения адреса динамически размещаемого элемента, а не самого элемента» [5].

Каждой динамической структуре данных сопоставляется статическая переменная типа указатель, ее значение – адрес этого объекта, посредством которой осуществляется доступ к динамической структуре.

Сами динамические величины не требуют описания в программе, поскольку во время компиляции память под них не выделяется. Во время компиляции память выделяется только под статические величины. Указатели – это статические величины, поэтому они требуют описания.

«Динамические структуры по определению характеризуются отсутствием физической смежности элементов структуры в памяти,

непостоянством и непредсказуемостью размера, числа элементов структуры в процессе ее обработки.

Поскольку элементы динамической структуры располагаются по непредсказуемым адресам памяти, адрес элемента такой структуры не может быть вычислен из адреса начального или предыдущего элемента. Для установления связи между элементами динамической структуры используются указатели, через которые устанавливаются явные связи между элементами. Такое представление данных в памяти называется свя́зным» [5].

Слайд 218

«Допустим, необходимо создать программу, позволяющую вводить данные о сотрудниках организации. Количество сотрудников неизвестно. Можно было бы создать массив записей с запасом. Однако, если данных о каждом сотруднике много, то каждая запись занимает много памяти. Получается, что мы будем расходовать много памяти впустую. Проблема решается путем построения цепочки взаимосвязанных структур» [3].

Каждая динамическая структура характеризуется

- взаимосвязью элементов;
- набором типовых операций над этой структурой.

Для организации связей между элементами динамической структуры каждый элемент структуры должен содержать информационную часть и указатель – ссылку **link [линк]** на следующий элемент. В динамические структуры можно объединять любую информацию. Особенность такой организации информации заключается в том, что память под хранение информации может выделяться и освобождаться непосредственно во время выполнения программы.

На слайде представлены операции с указателями при создании динамических структур данных.

Первая операция – описание переменной, указатель находится в неопределенном состоянии после его описания.

Вторая операция – выделение памяти. Используется оператор **new [нью]**. Данный оператор отводит место в оперативной памяти под хранение переменной указанного типа, в данном случае целого типа, и этот адрес вносится в указатель **p [пи]**.

Третья операция – указатель на пустой адрес, указатель никуда. Используется как признак конца списка.

Четвертая операция – оператор **delete [делет]** освобождает память, занимаемую динамической переменной, и память становится доступна для других динамических переменных. Указатель опять находится в неопределенном состоянии.

Слайд 219

Динамические структуры – это объединение информации в связанные динамические структуры:

- односвязные списки;
- двусвязные списки;
- бинарные деревья.

«Каждая компонента любой динамической структуры представляет собой структуру, содержащую, по крайней мере, два элемента: один – типа указатель, а второй – для размещения данных. В общем случае может содержать несколько элементов данных. Поле данных может быть переменной, массивом или структурой» [5].

Рассмотрим односвязные списки. На слайде приведено описание односвязного списка. Каждый элемент списка – это структура, содержащая информационную часть и указатель на следующий элемент структуры.

Каждый элемент односвязного списка имеет указатель, ссылку на следующий элемент списка. Тип указателя определяется через имя структуры. Имя структуры и есть пользовательский тип, базирующийся на стандартных типах данных языка **C++ [си плюс плюс]**.

Односвязные списки можно формировать:

– в виде очереди, когда новый элемент добавляется в конец очереди, а удаляется элемент из начала очереди;

– в виде стека, когда добавлять и удалять элемент можно только с одного конца списка, который называется вершиной стека.

Слайд 220

«Основной задачей при изучении динамических структур является изучение особенностей доступа к данным в динамических структурах, работы с памятью при использовании структур в программе.

В языке **C++ [си плюс плюс]** имеются средства создания динамических структур данных, которые позволяют во время выполнения программы образовывать объекты, выделять для них память, освобождать память, когда в них исчезает необходимость.

Динамические структуры данных – это любая структура данных, занимаемый объем памяти которой не является фиксированным. Размер подобной структуры ограничен только объемом оперативной памяти компьютера.

Если до начала работы с данными невозможно определить, сколько памяти потребуется для их хранения, память следует распределять во время выполнения программы по мере необходимости отдельными блоками. Блоки связываются друг с другом с помощью указателей. Такой способ организации данных называется динамической структурой данных, поскольку она размещается в динамической памяти и ее размер изменяется во время выполнения программы.

По сути, это очень похоже на обыкновенный массив, с той лишь разницей, что размер его не имеет ограничений и содержит, объединяет информацию различных типов» [5].

На слайде представлены основные операции над односвязными списками. Выполнение этих операций в массивах – процесс довольно трудоемкий.

Например, для удаления элемента массива придется перетаскивать элементы массива на место удаляемого. В односвязном списке эта процедура выполнится одной операцией – изменением ссылки предыдущего элемента на последующий, где текущий – это элемент, который надо удалить. Для добавления элемента в середину списка достаточно только перезамкнуть ссылки. Процесс сортировки можно реализовать непосредственно во время формирования динамической структуры.

Слайд 221

«Элемент динамической структуры в каждый момент может либо существовать, либо отсутствовать в памяти, поэтому его называют динамическим. Поскольку элементами динамической структуры являются динамические переменные, то единственным средством доступа к динамическим структурам и их элементам является указатель, то есть адрес,

на место их текущего расположения в памяти. Таким образом, доступ к динамическим данным выполняется специальным образом с помощью указателей» [5].

Указатель содержит адрес определенного объекта в динамической памяти. Адрес формируется из двух слов: адрес сегмента и смещение. Сам указатель является статическим объектом и расположен в сегменте данных.

«Для обращения к динамической структуре достаточно хранить в памяти адрес первого элемента структуры. Поскольку каждый элемент динамической структуры хранит адрес следующего за ним элемента, можно, двигаясь от начального элемента по адресам, получить доступ к любому элементу данной структуры» [4].

Теперь перейдем к конкретным задачам и рассмотрим создание динамических структур данных.

Слайд 222

Решим задачу, на примере которой подробно рассмотрим операции над односвязными списками. Каждый элемент списка должен содержать информацию о наименовании товара, количестве данного наименования товара и ссылку на следующий элемент списка. То есть каждый элемент списка представляет собой структуру.

Требуется составить программу, формирующую односвязный список в виде стека о товарах склада, и распечатать созданный список товаров.

Ввод прекращается, если вместо количества товаров ввести 0 [ноль].

При создании односвязного списка в виде стека необходимо определить два указателя: на вершину стека и на текущий элемент. Указатель на вершину стека – фактически единственная связь со списком. Значение

этого указателя нельзя терять, иначе потеряется связь со списком. Текущий указатель позволит перемещаться по списку от элемента к элементу.

На слайде представлен первый фрагмент программы. Описана структура с именем **Tovar [товар]**, состоящая из трех полей. Первые два поля – информационные, третье поле – указатель на следующий элемент динамической структуры, то есть указатель на следующий товар. Тип указателя должен совпадать с типом информации, на которую он указывает. Здесь **Tovar [товар]** – пользовательский тип данных.

Обратите внимание, что структура описана вне всяких функций и даже до команд препроцессора. Это вполне допустимо.

Слайд 223

В программе на слайде создана подпрограмма обхода и вывода на печать элементов односвязного списка, а именно – наименование и количество товара.

Обход элементов односвязного списка можно реализовать в цикле. В данной задаче используется рекурсивная функция обхода и вывода на печать информации. Рекурсивная функция с именем **output [аут пут]** представлена на слайде.

Данная рекурсивная функция не возвращает никакого результата, поэтому тип функции описан с помощью служебного слова **void [войд]**. В качестве формального параметра используется указатель на начало односвязного списка. Тип указателя – это тип описанной выше структуры.

Функция рекурсивно обращается сама к себе, пока не будет пройден весь список. При каждом рекурсивном обращении фактическим параметром является указатель на следующий элемент структуры.

Выход из процедуры осуществляется по достижении конца списка, то есть у последнего элемента структуры в поле «Указатель» ставится константа **NULL [нул]**.

«Доступ к данным в динамических структурах осуществляется с помощью операции стрелка, которую называют операцией косвенного выбора элемента структурного объекта, адресуемого указателем. Она обеспечивает доступ к элементу структуры через адресующий ее указатель того же структурного типа» [5].

Обратите внимание, что доступ к элементам структуры **name [нейм]** и **kol [кол]** осуществляется через указатель **p [пи]** и операцию косвенного выбора элемента структуры.

Слайд 224

На слайде представлена главная функция. Описаны два указателя:

- **top [топ]** – на вершину, или начало списка;
- **p [пи] –** на текущий элемент.

Первоначально указатель на начало списка устанавливается в неопределенное положение, константа **NULL [нул]**.

Организован цикл с постусловием для создания односвязного списка. Для каждого элемента односвязного списка с помощью оператора **new [нью]** выделяется динамическая память, и адрес вносится в указатель. Информация вносится с клавиатуры и как бы привязывается к односвязному списку. Ввод продолжается до тех пор, пока вместо количества товара будет введен ноль.

После завершения ввода идет обращение к рекурсивной функции **output [аут пут]**, которая распечатает информацию из созданного односвязного списка.

«Имея возможность явного манипулирования с указателями, которые могут располагаться как вне структуры, так и внутри отдельных ее элементов, можно создавать в памяти различные структуры.

Однако необходимо помнить, что работа с динамическими данными замедляет выполнение программы, поскольку доступ к величине происходит в два шага: сначала ищется указатель, затем по нему – величина» [3].

Слайд 225

Элемент двусвязного списка представляет собой структуру, имеющую информационную часть и два указателя.

«Для ускорения многих операций целесообразно применять переходы между элементами списка в обоих направлениях. В таком списке каждый элемент, кроме первого и последнего, связан с предыдущим и следующим за ним элементами.

Двусвязный список более гибкий. При включении элемента в список нужно использовать указатель как на элемент, за которым происходит включение, так и указатель на элемент, перед которым происходит включение. При исключении элемента из списка нужно использовать как указатель на сам исключаемый элемент, так и указатели на предшествующий или следующий за исключаемым элементы. Но так как элемент двусвязного списка имеет два указателя, то при выполнении операций включения или исключения элемента надо изменять больше связей, чем в односвязном списке.

Особое внимание следует обратить на то, что, в отличие от односвязного списка, здесь нет необходимости обеспечивать

позиционирование какого-либо указателя именно на первый элемент списка. Ведь благодаря двум указателям в элементах можно получить доступ к любому элементу списка из любого другого элемента, осуществляя переходы в прямом или обратном направлениях. Однако, по правилам хорошего тона программирования, указатель желательно ставить на заголовок списка» [5].

Перечисленные процедуры сводятся к переключению указателей между элементами структуры. Оперируя указателем **p [пи]** на предыдущий элемент списка и указателем **t [ти]** на последующий элемент, можно удалять, вставлять новый узел, всего лишь перезамкнув указатели.

Слайд 226

На слайде приведено описание двусвязного списка.

Имя структуры – это уникальный идентификатор. Каждый элемент структуры имеет информационную часть и две ссылки:

- **previous [привиэс] –** указатель на предыдущий элемент;
- **subsequent [саб си квэнт] –** указатель на последующий элемент.

Наличие ссылок на следующее звено и на предыдущее звено позволяет двигаться по списку от каждого звена в любом направлении: от звена к концу списка или от звена к началу списка, поэтому такой список называют двунаправленным.

В информационную часть могут входить переменные, массивы и даже структуры. Тип этих объектов определяет размер всей структуры.

Основные операции, осуществляемые с двусвязными списками:

- создание списка;
- печать списка;

– вставка элемента в список;
– удаление элемента из списка;
– поиск элемента в списке;
– проверка пустоты списка;
– удаление списка.

Операции для двунаправленного списка реализуются абсолютно аналогично соответствующим функциям для однонаправленного списка. Просматривать двунаправленный список можно в обоих направлениях.

Для того чтобы создать список, нужно создать сначала первый элемент списка, а затем при помощи функции добавить к нему остальные элементы. Добавление может выполняться как в начало списка, так и в конец списка.

Слайд 227

«Бинарное дерево является одним из важнейших и интересных частных случаев графа. Древовидная модель оказывается довольно эффективной для представления динамических данных с целью быстрого поиска информации.

Дерево – это структура данных, представляющая собой совокупность элементов и отношений, образующих иерархическую структуру этих элементов. Каждый элемент дерева называется вершиной или узлом дерева. Вершины дерева соединены направленными дугами, которые называют ветвями дерева. Начальный узел дерева называют корнем дерева, ему соответствует нулевой уровень. Листьями дерева называют вершины, в которые входит одна ветвь и не выходит ни одной ветви.

Все вершины, в которые входят ветви, исходящие из одной общей вершины, называются потомками, а сама вершина – предком» [5].

Бинарное дерево – это такая динамическая структура, в которой:

– есть только один узел, в который не входит ни одного ребра, который называется вершиной дерева;

– в каждый узел, кроме вершины, входит только одно ребро;

– из каждого узла исходит не более двух ребер.

При построении бинарного дерева используется следующее правило:

– от каждого узла влево вниз ставится меньший узел;

– от каждого узла вправо вниз ставится больший узел.

Преимуществом построения бинарных деревьев является быстрота поиска информации.

На слайде представлена последовательность чисел, с которыми построено бинарное дерево по перечисленным выше правилам. При описании бинарных деревьев каждый элемент структуры имеет два указателя: на элемент, стоящий вправо вниз, и на элемент, стоящий влево вниз.

Слайд 228

Рассмотрим описание структуры бинарное дерево и основные принципы построения бинарных деревьев.

Каждый элемент структуры имеет информационную часть и два указателя: на элемент, расположенный влево вниз от данного узла, и на элемент, расположенный вправо вниз от данного узла.

Основные операции над бинарными деревьями:

– переход от элемента к элементу;

– добавление нового элемента к списку;

– исключение элемента из списка;

– сортировка элементов.

При удалении элемента бинарного дерева руководствуются следующим правилом: надо от удаляемого узла сделать шаг влево и идти до конца вправо. Тогда бинарное дерево не рассыплется.

«Упорядоченное дерево – это дерево, у которого ветви, исходящие из каждой вершины, упорядочены по определенному критерию.

Деревья являются рекурсивными структурами, так как каждое поддерево также является деревом.

Действия с рекурсивными структурами удобнее всего описываются с помощью рекурсивных алгоритмов» [5].

Слайд 229

Тема 9. Файл данных (часть 1)

«Большинство компьютерных программ работают с файлами, и поэтому возникает необходимость создавать, удалять, записывать, читать, открывать файлы. Хранение данных через переменные или массивы является временным, до конца работы программы. Для долговременного хранения предназначены файлы, размещенные на внешних носителях.

Файл – именованный набор байтов, который может быть сохранен на некотором накопителе. Под файлом понимается некоторая последовательность байтов, которая имеет своё уникальное имя. В одной директории не могут находиться файлы с одинаковыми именами. Под именем файла понимается не только его название, но и расширение» [5].

Логически файл можно представить как цепочку байт, имеющих начало и конец. Работа с файлом предполагает подключение заголовочного файла **fstream [эф стрим]** из стандартной библиотеки**.**

Для записи в файл его необходимо открыть. Функция **ofstream [оф стрим]** открывает файл для записи.

При обращении к файлу используют файловую переменную. Файловая переменная – это уникальный идентификатор, строится по всем правилам построения идентификаторов на языке **C++ [си плюс плюс]**.

Указывается также имя файла и режим доступа к файлу. Синтаксис языка требует заключить имя файла в двойные кавычки. Если файл не находится в одной папке с программным кодом, то прописывают путь доступа к файлу.

Запись в файл определяется положением внутреннего указателя. Положение указателя меняется либо автоматически, либо за счет явного управления его положением. Положение внутреннего указателя определяется режимом открытия файла. При каждом обращении к файлу, то есть при записи в файл, внутренний указатель автоматически перемещается в конец очередной порции информации, записанной в файл.

Файловая переменная, связанная с именем файла, фактически идентифицирует файл. Далее в программе имя файла не используется. Обращение к файлу осуществляется через файловую переменную. При повторном обращении к файлу, например при чтении из файла, в пределах одной программы нельзя использовать одну и ту же файловую переменную.

На слайде приведен пример, демонстрирующий открытие файла для записи.

Слайд 230

Работа с файлами предполагает следующие операции:

– создание файла;
– запись в файл;
– чтение из файла;
– добавление информации, или дозапись, в уже существующий файл.

Рассмотрим процедуру чтения из файла. Прежде всего необходимо открыть файл для чтения. На слайде представлены процедуры открытия файла для чтения. С помощью функции **ifstream [иф стрим]** файловую переменную связывают с именем файла.

Рассмотрим функцию, которая открывает файл для чтения. Указывается файловая переменная, которая в дальнейшем идентифицирует файл в функциях обработки.

Как уже говорилось ранее, в файле есть внутренний указатель. Чтение из файла определяется положением внутреннего указателя, который может меняться либо автоматически, либо за счет явного управления его положением. При каждом чтении из файла внутренний указатель перемещается на начало следующей компоненты файла.

Компонента файла – это та порция информации, которая была записана в файл при его создании. В файле можно хранить любую информацию. Файлы позволяют хранить информацию разного типа, объединяя ее в структуры, то есть создавать файлы структур. В следующих разделах мы подробно рассмотрим файлы структур.

Положение внутреннего указателя в файле зависит от режимов открытия файла. Если режим не указан, как в примере на слайде, то при открытии файла для чтения внутренний указатель автоматически устанавливается в начало файла.

Использование функции открытия файла для чтения также предполагает подключение заголовочного файла **fstream [эф стрим]**.

Слайд 231

При работе с файлами может возникнуть ситуация, когда в пределах одной программы необходимо и записывать информацию в файл, и

читать информацию из файла. Подробно остановимся на режимах открытия файла для записи и чтения в пределах одной программы.

Рассмотрим два режима записи в файл.

Первый режим **ate [эйт]** используется для создания нового файла, указатель устанавливается в начало файла. Если указать этот режим для уже существующего файла, можно потерять информацию, хранящуюся в файле.

Указатель установится в начало файла и при записи старая информация, как говорится, затирается.

Второй режим **out [аут]** используется для добавления информации – дозаписи – в уже существующий файл, внутренний указатель устанавливается в конец файла.

В режиме чтения из файла **in [ин]** внутренний указатель устанавливается в начало файла. Этот режим можно опустить, не указывать, он устанавливается автоматически.

Слайд 232

«Когда вы создаёте файловую переменную, связываемую с файлом по имени или полному пути файла, вы задаёте этой переменной тип, который может быть или **ifstream [иф стрим]**, или **ofstream [оф стрим]**, или **fstream [эф стрим]**. Вот этот тип называют файловым потоком. В каком режиме окажется файл, если режим не указан явно, зависит от выбранного типа. Если выбран тип чтение файла, то по умолчанию файл будет открываться для чтения, если выбран тип запись в файл, то по умолчанию файл будет открываться для записи в файл. Есть ещё режим, который получается при комбинации двух режимов – режима чтения и режима записи в файл. В таком режиме файл по умолчанию будет открыт и для чтения, и для записи сразу» [6].

Для работы с файлами в программе прежде всего надо открыть файл. На слайде продемонстрирована функция **fstream [эф стрим]**, позволяющая открыть файл для записи и чтения.

В функции открытия файла можно указать режимы работы.

Первый режим **ate [эйт]** – запись в файл, указатель автоматически устанавливается в конец файла.

Второй режим **out [аут]** – указатель устанавливается в начало файла.

Использование функции открытия файла для чтения и записи также предполагает подключение заголовочного файла **fstream [эф стрим]**.

На слайде приведен пример открытия файла для записи и чтения. Несколько режимов можно объединять между собой. Объединение режимов достигается путём использования операции дизъюнкция, эквивалентной логическому ИЛИ. Знак этой операции ставится между режимами.

Здесь **ff [эф эф]** – файловая переменная, через которую осуществляется доступ к файлу, то есть запись в файл или чтение из файла. Файловая переменная связана с именем конкретного файла. При указании имени файла можно также указать путь доступа к файлу, если файл не находится в текущей папке.

Слайд 233

Работа с файлами предполагает проверку на существование файла как при записи в файл, так и при чтении из файла. На слайде приведен пример, демонстрирующий проверку на существование файла, где указывается файловая переменная, которая была связана с именем переменной при открытии. Используя операции открытия файла, желательно осуществлять проверку на существование файла.

В логическом операторе **if [иф]** анализируется файловая переменная **ff [эф эф]**. Логическое выражение может принимать одно из двух значений: **true [тру]** или **false [фолс]**.

Если файл существует, то есть открытие файла прошло удачно, значение указанного условия будет истинным. Если файл не существует или не найден по указанному пути доступа к файлу, указанное логическое выражение выработает значение «ложь». На экран выведется сообщение об ошибке, и функция **exit [екзит]** прекратит выполнение программы.

Используя файлы данных, во избежание потери информации перед завершением работы программы желательно закрыть файл.

На слайде представлена функция **close [клоз]**, обеспечивающая закрытие файла. При обращении к этой функции указывается файловая переменная, связанная с именем файла при открытии. Необходимо указывать пустые круглые скобки рядом с именем функции, этого требует синтаксис языка **C++ [си плюс плюс]**.

Слайд 234

По способу доступа к информации файлы делятся на файлы последовательного доступа и файлы произвольного доступа.

Файлы последовательного доступа – это файлы, хранящие информацию в неструктурированном виде. Поиск в таких файлах осуществляется последовательным считыванием информации из файла. Так и обращение к определённому участку файла каждый раз требует выполнить чтение с начала файла. Файл – это цепочка логически связанных байт. Как в свое время информация была записана в файл, так же она и будет считываться из файла.

На слайде представлены процедуры записи и чтения одной компоненты в файлах последовательного доступа.

И запись, и чтение осуществляются через некоторую переменную **name [нейм]**. Тип этой переменной, то есть размер компонент файла, определяет количество байт, которые записываются в файл или считываются из файла.

При записи информации в файл после записи каждой компоненты вносится константа **\n [слеш эн]**. Это обеспечивает порцию считывания из файла. При каждом обращении к файлу считывается одна компонента файла.

Обратите внимание, что запись в файл и чтение из файла аналогичны организации потоков ввода-вывода информации. Работу этих функций обеспечивает стандартная библиотека **fstream [эф стрим]**.

Слайд 235

На слайде представлен фрагмент создания файла последовательного доступа, содержащего целые числа.

Открыт файл для записи, указаны файловая переменная **f_file [эф файл]** и имя файла. Имя файла, путь доступа к файлу указываются в двойных кавычках. Далее в программе обращение к файлу будет происходить только через файловую переменную.

Для записи в файл выделена переменная **buf [буф]** целого типа. Тип этой переменной определяет величину каждой компоненты файла, то есть порции записи в файл в байтах.

Переменная **n [эн]** определяет количество чисел, которые будут введены с клавиатуры и через буфер записываются в файл. Ввод осуществляется в цикле. Каждое введенное число на каждом шаге итерации цикла записывается в файл.

При завершении ввода чисел и записи в файл внутренний указатель находится в конце файла. Исходный файл необходимо закрыть. При повторном открытии файла для чтения указатель автоматически перейдет на начало файла.

Сформировать из созданного файла целых чисел два файла, один – с четными числами исходного файла, другой – с нечетными числами. На следующем слайде приведен фрагмент программы решения этой задачи.

Слайд 236

В представленном на слайде фрагменте открыты три файла:

– **ff_file [эф эф файл]** – для чтения из файла исходных чисел;

– **fc_file [эф си файл]** – для создания файла четных чисел;

– **fn_file [эф эн файл]** – для создания файла нечетных чисел.

Для открытия исходного файла используется новая файловая переменная. На предыдущем слайде при открытии файла для записи использовалась файловая переменная **f_file [эф файл]**.

Чтение из исходного файла реализовано в структуре оператора цикла с предусловием. Условием продолжения цикла является процедура чтения из файла. Это условие трактуется как «пока читается из файла».

Каждое число, прочитанное из исходного файла, анализируется на четность и записывается либо в файл четных чисел, либо в файл нечетных чисел.

Все три файла необходимо закрыть. При закрытии каждого файла указывается своя файловая переменная.

Слайд 237

Не забывайте о заголовочном файле, который необходимо подключить для реализации процедур обработки файлов.

Заранее неизвестно, сколько чисел попало в файл четных чисел и сколько в файл нечетных чисел. Поэтому для чтения информации из файлов используется оператор цикла с предусловием. В файлах при создании автоматически формируется признак конца файла. Цикл продолжается до тех пор, пока читается из файла, пока не встретился признак конца файла.

В цикле с предусловием читается и выводится на экран очередная компонента файла. После каждой процедуры чтения из файла внутренний указатель переносится в начало следующей компоненты файла.

Процедура последовательно выполняется сначала для файла четных чисел, а затем для файла нечетных чисел. Выводу на печать содержимого созданных файлов предшествует сообщение на экран о содержимом файла.

Каждая компонента файла считывается в переменную с именем **buf [буф]** и выводится на печать.

Функция чтения из файла вставлена как условие продолжения цикла в операторе **while [вайл]**.

Слайд 238

Далее рассмотрим работу с файлами, каждая компонента которых является структурой, содержащей информацию разного типа. Файлы структур могут быть как последовательного, так и произвольного доступа. Рассмотрим создание файлов структур последовательного доступа. На слайде приведен пример описания структуры, то есть описание каждой компоненты файла. Размер каждой компоненты зависит от элементов описанной структуры.

Рассмотрим задачу: нужно создать файл последовательного доступа, содержащий информацию о товарах некоторого склада.

Информация о каждом товаре содержит:

– наименование товара;

– количество товаров каждого наименования;

– цену единицы товара.

Требуется распечатать информацию из созданного файла и подсчитать общую стоимость товаров на складе.

Определен пользовательский тип данных с именем **Tovar [товар]**.

Для записи и чтения из файла необходимо ввести переменную-буфер, размер которой, то есть количество байт, определено описанной структурой.

Слайд 239

На слайде представлена программа создания файла последовательного доступа, каждая компонента которого имеет описанную структуру. Рассмотрим первый фрагмент программы решения этой задачи.

Подключены заголовочные файлы, обеспечивающие работу функций обработки файлов.

В данной программе используется препроцессорная команда **define [дэ файн]**, которая задаёт символическую константу. Символическая константа связывается с именем файла. При необходимости можно указать путь доступа к файлу. Имя файла задается в двойных кавычках. Далее в программе эта символическая константа используется вместо имени файла.

Описана структура, содержащая:

– символьный массив под хранение наименования товара;

– цену товара – переменную целого типа;

– количество товаров – переменная вещественного типа.

Создан пользовательский тип – **Tovar [товар]**.

Слайд 240

Для обращения к элементам структуры введена переменная **buf [буф]**.

С помощью функции **ofstream [оф стрим]** открывается файл последовательного доступа для записи.

С клавиатуры вводится количество наименований товаров. В цикле вводится информация о каждом наименовании товаров. Обращение к

отдельным элементам структуры осуществляется с помощью переменной буфер.

Каждая единица информации записывается в файл и завершается, или, как говорят, подпирается символом **\n [слеш эн]**.

Это позволит в дальнейшем при чтении из файла получить информацию согласно описанной выше структуре: наименование, цена, количество. При чтении из файла необходимо соблюдать этот же порядок. Как говорилось выше, файл – это последовательность байт. Если при чтении из файла нарушить последовательность, информация считается из файла, но будет нечитабельна.

Обратите внимание, как происходит обращение к отдельным элементам структуры – через имя переменной, имеющей тип структура и имя конкретного поля. Между ними ставится точка.

По окончании процедуры записи в файл внутренний указатель находится в конце файла.

Закрываем файл, чтобы открыть его для чтения.

Слайд 241

Следующий фрагмент программы демонстрирует чтение из файла и подсчет общей стоимости товаров на складе. Для подсчета общей стоимости товаров определена переменная **s [эс]**. С помощью функции **ifstream [иф стрим] о**ткрывается файл для чтения. Используется новая файловая переменная **ff [эф эф]** для обращения к файлу. Вместо имени файла используется символьная константа.

При открытии файла для чтения внутренний указатель устанавливается автоматически в начало файла.

Чтение информации из файла выполняется в цикле, количество циклов определено введенным значением переменной **n [эн]**. При каждом

обращению к файлу считывается единица информации из файла. В файлах последовательного доступа нельзя считать сразу всю структуру, всю информацию об одном наименовании товара. Надо помнить, как в свое время был создан файл, то есть записывалась информацию в файл. В такой же последовательности надо считывать информацию из файла. И если не соблюдать перечисленных выше правил, информация будет считана, но не адекватно внесенной в файл информации.

Для подсчета общей стоимости товаров на складе определена вещественная переменная, которой было присвоено значение ноль в начале программы.

В конце программы необходимо закрыть файл с помощью функции **close [клоз].** Посчитанная общая стоимость товаров на складе выводится на печать.

Если не указан путь доступа к файлу при его создании, файл будет создан в текущей папке, где и код программы.

Слайд 242

Перейдем к рассмотрению файлов произвольного доступа. Файлы последовательного доступа выигрывают у файлов с произвольным доступом по компактности, но проигрывают по скорости доступа к информации.

Еще раз напомним, что файлы последовательного доступа – это последовательность неструктурированных байт. Файлы с последовательным доступом читаются от начала к концу, поэтому невозможно одновременно и считывать из них данные, и записывать таковые. Чтобы изменить одну запись файла последовательного доступа, его нужно весь записать заново.

Если требуется частый доступ к данным, хранящимся в некотором файле, следует использовать файлы с произвольным доступом.

Файлы с произвольным доступом – файлы, хранящие информацию в структурированном виде. Файлы произвольного доступа, так же как и файлы последовательного доступа, должны быть открыты или для записи, или для чтения, или для записи и чтения одновременно.

Запись в файл произвольного доступа реализует функция – **write [райт]**, а позиционирование – функция **seekp [сик пут]**, где слово **put [пут]** означает «положить».

Работа этих функций означает: установить внутренний указатель на **n [энный]** байт от начала файла и записать значение переменной **buf [буф]** с размером **sizeof [сайс оф]** байт, начиная с этой позиции вперед.

Здесь **ff [эф эф]** – файловая переменная, которая связана с именем конкретного файла.

Слайд 243

Выше рассматривались задачи обработки файлов последовательного доступа, содержащие информацию разного типа, объединенную в структуру.

Каждый элемент структуры последовательно записывался и последовательно считывался из файла.

Еще одним преимуществом файлов произвольного доступа, помимо позиционирования внутреннего указателя, является возможность записи и считывания из файла всей структуры одним обращением к файлу.

При обработке файлов надо помнить, что как в свое время информация была записана в файл, так же и надо ее считывать, такими же порциями.

Если появляется необходимость вносить изменения в файл, то это сложно осуществить в файлах последовательного доступа. Управлять

внутренним указателем в файлах произвольного доступа можно с помощью функций, представленных на слайде.

Слайд 244

Чтение из файлов произвольного доступа реализует функция **read [рид]**, а позиционирование – функция **seekg [сик гет]**, где слово **get [гет]** означает «взять».

Работа этих функций означает: установить внутренний указатель на **n [энный]** байт от начала файла и присвоить переменной **buf [буф]**, то есть считать, значение, записанное от позиции **n [эн]** на **sizeof [сайс оф]** байт вперед, где **ff [эф эф]** – файловая переменная.

Поиск в файлах произвольного доступа осуществляется в области адресов и завершается обращением непосредственно к искомому участку. Дисковое пространство, занимаемое таким файлом, поделено на одинаковые участки, имеющие одинаковую структуру полей.

Напомним, что операция **sizeof [сайс оф]** определяет объем памяти в байтах.

После каждой процедуры записи или чтения в файлах произвольного доступа внутренний указатель перемещается на начало следующей компоненты файла. Если необходимо читать компоненты файла подряд, то функцию позиционирования можно опустить.

Преимущество файлов произвольного типа заключается в том, что если создать файл структур, то можно и записать в файл, и считать из файла всю структуру целиком.

Слайд 245

Рассмотрим задачу: нужно создать файл целых чисел, из исходного файла сформировать новый, содержащий числа исходного файла, стоящие на четных местах.

Из условия задачи видно, что необходимо позиционирование внутреннего указателя для перебора чисел на четных местах. Для решения этой задачи создается файл произвольного доступа. Как известно, в файлах произвольного доступа есть возможность управлять внутренним указателем. Именно с положения внутреннего указателя происходит запись в файл и чтение из файла.

Подключены заголовочные файлы **fstream [эф стрим]** для применения функций обработки файлов и **iostream [ай о стрим]** для организации ввода-вывода информации.

С помощью файловой переменной **f_file [эф файл**] открыт доступ к файлу для записи. Введена переменная буфер, тип которой, согласно условию задачи, целый, то есть имеет тип компонент файла. Переменная **n [эн]** – это количество чисел, которые необходимо записать в файл.

Слайд 246

На слайде продемонстрирован фрагмент программы, создающий исходный файл целых чисел.

С клавиатуры вводится количество чисел **n [эн]**. В цикле вводятся целые числа с клавиатуры, и каждое введенное число записывается в файл. Нет необходимости в позиционировании внутреннего указателя в процессе создания нового файла. После записи очередной компоненты в файл произвольного доступа внутренний указатель автоматически устанавливается в начало следующей компоненты.

Размер каждой компоненты файла определяется типом переменной – буфера.

Исходный файл необходимо закрыть.

Далее открываются два файла:

– исходный файл для чтения;

– новый файл для записи, содержащий числа исходного файла, стоящие на четных местах. Имя файла указано в двойных кавычках.

Обратите внимание, что при повторном открытии исходного файла используют новую файловую переменную **ff [эф эф]**.

Слайд 247

На слайде следующий фрагмент программы демонстрирует создание нового файла.

В цикле считывается информация из исходного файла с помощью функции **read [рид]** и записывается в новый файл. В законе изменения параметра цикла шаг изменения равен двум, что позволяет, благодаря указанной формуле, позиционировать внутренний указатель в исходном файле. Позиционирование осуществляет функция **seekp [сик пут]**.

Далее необходимо закрыть оба файла. Созданный файл открывается для чтения с указанием новой файловой переменной. В цикле с предусловием информация читается из файла и выводится на экран. Обратите внимание, что функция чтения из файла **read [рид]** одновременно является условием продолжения цикла, то есть пока читается из файла.

Обращение к файлам, то есть процедура чтения из файла и процедура записи в файл, осуществляется с помощью переменной **buf [буф]**.

Слайд 248

На слайде подробно рассмотрен расчет позиционирования внутреннего указателя, так как из исходного файла необходимо считывать только числа, стоящие на четных местах. В различных системах количество байт под тот или иной тип данных может отличаться. На слайде приведен расчет для целочисленных переменных, занимающих четыре байта памяти. С помощью функции **sizeof [сайс оф]** можно определить количество байт, отводимых под переменные разного типа.

Расчет зависит от размера компонент файла, в данной задаче это четыре байта, то есть размер переменных целого типа.

По указанной формуле рассчитывается байт, на который устанавливается внутренний указатель исходного файла, и производится чтение очередной компоненты.

Расчет позиционирования внутреннего указателя всегда связан с размером компонент файла, то есть с количеством байт, которые необходимо считать из файла.

Слайд 249

Тема 9. Файл данных (часть 2)

Рассмотрим задачу. Нужно создать файл, содержащий информацию о товарах некоторого склада. Каждая компонента файла – структура, содержащая информацию о товарах некоторого склада: наименование товаров, количество каждого наименования товаров и цена единицы товара.

На слайде приведена структура, определяющая размер компонент файла, определяющая количество байт.

Переменная **buf [буф]** имеет пользовательский тип структура. Размер этой переменной определен элементами структуры.

Переменная **buf [буф]** описана непосредственно после определения структуры. В принципе, имя этой структуры при описании структуры можно опустить. Именно через переменную **buf [буф]** будет происходить обмен информацией с файлом.

Слайд 250

С клавиатуры вводится количество наименований товаров, то есть значение переменной **n [эн]**.

Открывается файл для записи. Ввод информации осуществляется в цикле с параметром.

На каждой итерации цикла с клавиатуры вводится информация об одном наименовании товара, отдельно – наименование товара, цена и количество.

Функция **write [райт]** позволяет одним обращением всю структуру записать в файл. Размер компонент файла вычисляется автоматически операцией **sizeof [сайс оф]**.

В данном случае нет необходимости в позиционировании внутреннего указателя файла.

По окончании процедуры записи в файл внутренний указатель находится в конце файла. Файл был открыт для записи.

Слайд 251

Для того чтобы просмотреть содержимое созданного файла, открываем файл для чтения.

Чтение из файла осуществляется в цикле с предусловием.

В качестве условия выполнения цикла в операторе **while [вайл]** используется функция чтения из файла.

Каждое обращение к файлу соответствует чтению одной компоненты файла, равной размеру описанной выше структуры.

Каждый элемент структуры через переменную **buf [буф]** выводится на экран.

Перед концом выполнения программы необходимо закрыть файл во избежание потери или порчи информации.

Файл, созданный этой программой, мы будем использовать в следующей программе.

Слайд 252

В данной задаче будем использовать базу данных, то есть файл, созданный в предыдущей программе.

Необходимо подсчитать общую стоимость товаров определенного наименования в базе данных, в файле.

При обращении к файлу надо точно знать, как он был в свое время создан, то есть структуру каждой компоненты файла.

Описана структура, которая должна совпадать со структурой, описанной в предыдущей задаче.

Введена переменная **s [эс]** для подсчета стоимости определенного товара в файле, значение которой надо подготовить – обнулить.

Для поиска по наименованию определенного товара в файле введена переменная **name [нейм]**, одномерный массив символов. Значение этой переменной вводится с клавиатуры.

Слайд 253

Отрывается доступ к файлу, созданному в предыдущей программе. Можно указать путь доступа к файлу, если файл не в текущей папке.

Чтение информации из файла выполняется в цикле с предусловием. Чтение из файла, функция **write [райт]**, используется как условие продолжения цикла, то есть пока читается из файла.

Каждое обращение к файлу соответствует считыванию одной компоненты, размер которой, то есть количество считанных из файла байт, соответствует размеру описанной в начале программы структуре. Наименование структуры, имена переменных, которые являются элементами структуры, могут не совпадать с описанием структуры при создании файла. Но типы элементов структуры и последовательность элементов необходимо сохранить.

Считывается информация из файла в переменную-буфер **buf [буф]** и сравнивается с искомым в файле товаром. Функция **strcmp [стрим си эм пи]** сравнивает побайтно две переменные, искомое наименование товара и наименование товара из файла. Если переменные совпали, функция вырабатывает значение ноль. Информация выводится на печать, одновременно подсчитывается общая стоимость товаров на складе данного наименования. Данную функцию сравнения символьных переменных мы подробно ранее рассматривали в нашем курсе. В конце вычислений необходимо закрыть файл.

Слайд 254

В задаче на слайде подразумевается внесение изменений в уже существующий файл.

Наименование товара, стоимость которого надо изменить, вводится с клавиатуры.

Для решения этой задачи необходимо точно знать, как файл был в свое время создан, то есть знать структуру файла, последовательность

расположения элементов структуры. Размер каждой компоненты определяется типом элементов структуры.

Задача предполагает чтение очередной компоненты файла и, если это искомый товар, изменение стоимости, а также запись измененной информации в файл на старое место. Это возможно, если применить функции управления внутренним указателем в файлах произвольного доступа, которые были рассмотрены выше.

Файл произвольного доступа позволяет позиционировать внутренний указатель, что и позволит изменять информацию, не создавая новый файл.

Слайд 255

На слайде представлен фрагмент программы.

С помощью функции **fstream [эф стрим]** открыт доступ к файлу для записи и чтения. Работа этой функции рассматривалась выше. Происходит одновременно и чтение из файла, и запись в файл с использованием одной и той же файловой переменной.

Описан пользовательский тип **Tovar [товар]**, который определяет структуру и размер компонент файла.

Переменная **buf [буф]** будет использована при обращении к файлу, при записи или чтении. Переменная **n [эн]** – общее количество различных наименований товаров в файле.

С клавиатуры вводится наименование товара, стоимость которого надо изменить.

Слайд 256

На слайде представлен следующий фрагмент программы. Комментарии помогут подробно рассмотреть этот фрагмент.

Использован цикл с параметром, где параметр цикла используется при позиционировании внутреннего указателя в файле.

Рассмотрим фрагмент подробно:

– внутренний указатель устанавливается на начало компоненты файла с номером **i [ай]**;

– считывается очередная компонента файла;

– сравнивается наименование товара в очередной компоненте файла с наименованием искомого товара;

– если условие выполнилось, изменяется стоимость данного товара;

– указатель возвращается назад, то есть на начало этой же компоненты;

– измененная информация записывается на старое место.

Обратите внимание, что изменяется только часть информации компоненты файла, а именно – цена. Наименование товара и количество остаются неизменными. Записать в файл необходимо всю структуру.

Слайд 257

Откроем файл только для чтения. Для открытия файла введена новая файловая переменная **tt [ти ти]**.

В цикле с предусловием читаем информацию из файла и выводим на экран. Обратите внимание на условие продолжения цикла. Этим условием является функция чтения из файла. Условие трактуется так – пока читается из файла, то есть пока не будут считаны все компоненты файла. Естественно, в файле присутствует признак конца файла.

Размер каждой считанной компоненты определяется операцией **sizeof [сайс оф]**. Считывается вся компонента из файла, и каждый элемент структуры выводится на печать через переменную **buf [буф]**. Указываются

имя структуры и через точку имя конкретного элемента структуры, а именно – наименование товара и цена.

Во избежание потери информации перед выходом из программы необходимо закрыть файл.

Слайд 258

«Файл – это именованная область внешней памяти, в которой хранится логически завершенный объем данных. Текстовый файл – это файл, в котором каждый символ из используемого набора символов хранится в виде одного байта – кода, соответствующего символу. Текстовые файлы разбиваются на несколько строк с помощью специального символа – конец строки.

Поток – это абстрактное понятие, относящееся к любому переносу данных от источника к приемнику.

Функции библиотеки ввода-вывода языка С++ [си плюс плюс], поддерживающие обмен данными с файлами на уровне потока, позволяют обрабатывать данные различных размеров и форматов, обеспечивая при этом буферизованный ввод и вывод. Таким образом, поток представляет собой этот файл вместе с предоставленными средствами буферизации. Чтение данных из потока называется извлечением, вывод в поток – помещением, или включением. Поток определяется как последовательность байтов и не зависит от конкретного устройства, с которым производится обмен, – оперативной памяти, файла на диске, клавиатуры или принтера. Обмен с потоком для увеличения скорости передачи данных производится, как правило, через специальную область оперативной памяти – буфер. Буфер накапливает байты, и фактическая передача данных выполняется после заполнения буфера. При вводе это дает возможность исправить ошибки, если данные из буфера еще не отправлены в программу.

Когда программа начинает выполняться, автоматически открываются следующие потоки:

– стандартный поток ввода;

– стандартный поток вывода;

– стандартный поток вывода сообщений об ошибках.

По умолчанию стандартному потоку ввода ставится в соответствие клавиатура, а потокам вывода соответствует экран монитора» [5].

Слайд 259

На слайде приведены основные функции при работе с текстовыми файлами в режиме записи в файл.

«Существуют два основных типа файлов: текстовые и двоичные. В предыдущих темах мы уже познакомились с двоичными файлами.

К текстовым файлам прибегают в тех случаях, когда возникает необходимость обрабатывать большие объемы данных, не вводя их с клавиатуры.

Текстовыми файлы состоят из любых символов. Они организуются по строкам, каждая из которых заканчивается символом конца строки.

В конце текстового файла присутствует признак конца файла.

С помощью любого текстового редактора можно просмотреть содержимое текстового файла. При записи информации в текстовый файл все данные преобразуются к символьному типу и хранятся в символьном виде.

В программах на языке **C++ [си плюс плюс]** при работе с текстовыми файлами необходимо подключать библиотеки **iostream [ай о стрим]** и **fstream [эф стрим]**.

Для того чтобы записывать данные в текстовый файл, необходимо:

– описать файловую переменную типа **ofstream [оф стрим]**;

– открыть файл с помощью функции **open [опэн]**;

– вывести информацию в файл;

– обязательно закрыть файл.

В функции записи в текстовый файл указывается имя файла и режим открытия файла для записи.

После открытия файла в режиме записи в него можно писать точно так же, как и при выводе на экран, только вместо стандартного устройства вывода **cout [си аут]** необходимо указать имя открытого файла. Имя файла определяется через файловую переменную» [5].

Открытые на диске файлы после окончания работы с ними рекомендуется закрыть. После окончания работы файлы закрываются функцией **close [клоз]** во избежание потери информации в файле.

Слайд 260

На слайде приведены основные функции при работе с текстовыми файлами в режиме чтения из файла.

«Для того чтобы считать данные из текстового файла, необходимо:

– описать файловую переменную типа **ifstream [иф стрим]**;

– открыть файл с помощью функции **open [опэн]**;

– считать информацию из файла, при считывании каждой порции данных необходимо проверять, достигнут ли конец файла;

– закрыть файл.

После открытия файла в режиме чтения из него можно считывать информацию точно так же, как и с клавиатуры. Вместо функции **cin [син]** нужно указать имя потока, определенного через файловую переменную, из которого будет происходить чтение данных» [5].

На слайде приведены основные функции описания файловой переменной и открытие файла для чтения, а также представлен пример чтения информации из текстового файла.

Например, представлено чтение из потока **ff [эф эф]** в переменную **a [эй]**. Параметр, указывающий режим открытия файла, может отсутствовать, в этом случае файл открывается в режиме по умолчанию для данного потока.

Указывая имя файла, можно указать путь доступа к файлу. В программах на языке **C++ [си плюс плюс]** при работе с текстовыми файлами в потоковом режиме необходимо подключать библиотеки **iostream [ай о стрим]** и **ofstream [оф стрим]**.

После удачного открытия файла в любом режиме в файловой переменной **ff [эф эф]** будет храниться значение **true [тру]** – истина, в противном случае – **false [фолс]**, ложь. Это позволит проверить корректность операции открытия файла.

Слайд 261

На слайде приведены три примера:

- открытие файла для записи;
- открытие файла для чтения;
- открытие файла для записи и чтения.

Если в пределах одной программы возникает необходимость и записывать информацию в файл, и считывать информацию из файла, используют третий подход.

При открытии файла впервые, то есть при создании нового файла, в файле автоматически создается признак конца файла. При дозаписи в уже существующий файл признак конца файла автоматически отодвигается.

«После удачного открытия файла в любом режиме в файловой переменной **ff [эф эф]** будет храниться единица, то есть **true [тру]**, в противном случае – ноль, то есть **false [фолс]**. Это позволит проверять корректность операции открытия файла.

Чтение из файла выполняется корректно до достижения конца файла. Поток, как тип данных, – это тип, который описывает объекты, реализующие процесс ввода и вывода данных в файлы.

Режимы открытия файлов – это наборы допустимых действий с открываемыми в программе файлами.

После открытия файла в режиме записи будет создан пустой файл, в который можно записывать информацию. Если необходимо открыть существующий файл и при этом сохранить его содержимое, следует использовать определенный режим» [6].

Слайд 262

«Два числа в текстовом редакторе считаются разделенными, если между ними есть хотя бы один из символов: пробел, табуляция, символ конца строки. Хорошо, когда заранее известно, сколько и какие значения хранятся в текстовом файле.

Однако часто известен лишь тип значений, хранящихся в файле, при этом их количество может быть различным. Для решения данной проблемы

необходимо считывать значения из файла поочередно, а перед каждым считыванием проверять, достигнут ли конец файла» [5].

Для определения конца файла используют функцию **eof [энд оф файл]**.

На слайде приведен фрагмент чтения информации из текстового файла в структуре оператора **while [вайл]**. Цикл, то есть чтение из файла, продолжается до тех пор, пока не достигнут конец файла.

Файловая переменная **ff [эф эф]** может принимать одно из двух значений: **true [тру]** или **false [фолс]**, в зависимости от того, достигнут ли конец файла.

«Итак, подведем итоги.

На языке **C++ [си плюс плюс]** определен специальный тип данных – поток.

Каждый из потоков: **fstream [эф стрим]**, **ifstream [иф стрим]**, **ofstream [оф стрим]** – служит для работы с файлами в определенном режиме.

Прототипы функций по работе с файлами в потоковом режиме находятся в стандартных библиотеках.

Перед началом работы с файлом его необходимо открыть, указав режим открытия.

Любой открытый в программе файл необходимо закрыть после использования» [5].

Слайд 263

Рассмотрим режимы записи и чтения из текстовых файлов.

Рассмотрим режимы записи.

Первый режим – открыть файл в режиме записи данных, при этом информация в существующем файле уничтожается. Этот режим устанавливается по умолчанию для потоков **ofstream [оф стрим]**.

Второй режим – открыть файл в режиме записи данных в конец файла.

Третий режим – передвинуть внутренний указатель в конец уже открытого файла.

Четвертый режим – очистить файл, действия, аналогичные первому режиму.

Режим чтения из файла: открыть файл в режиме чтения данных. Режим является режимом по умолчанию для потоков **ifstream [иф стрим]**.

Режимы, которые устанавливаются по умолчанию, можно не указывать.

Слайд 264

Рассмотрим пример обработки текстовых файлов.

«В программах зачастую необходимо обрабатывать данные больших объемов, причем сами данные и результаты обработки требуется сохранять относительно долгое время. Проблему выделения дополнительных ресурсов для хранения обрабатываемых данных можно решить за счет динамической памяти, однако ее ресурс ограничен. Организовать длительное хранение информации только средствами оперативной памяти практически невозможно ввиду энергозависимости устройства. Поэтому работу с объемными данными и организацию долговременного хранения данных в языках программирования осуществляют с помощью файлов, расположенных на внешних носителях. При этом под внешними устройствами следует понимать устройства ввода-вывода данных, к которым также можно отнести и файлы. Прототипы основных функций для работы с файлами входят в стандартную библиотеку» [6].

Создадим файл вещественных чисел. Числа вводятся с клавиатуры и записываются в файл.

Подключены заголовочные файлы:

– **iostream [ай о стрим]** – для ввода информации с клавиатуры и вывода на экран;

– **fstream [эф стрим]** – для обращения к функциям обработки файла;

– **conio [конио]** – для работы функции **gech [геч]**.

Введены переменные:

– **i [ай]** – параметр цикла;

– **n [эн]** – количество чисел, которые надо записать в файл;

– **buf [буф]** – переменная-буфер, через которую числа записываются в файл, тип которой определяет количество байт записываемых в файл;

– **f [эф]** – файловая переменная, связанная с именем файла для записи в файл.

Слайд 265

На слайде представлен следующий фрагмент программы. С клавиатуры вводится количество чисел **n [эн]**, которые необходимо записать в файл.

Организован цикл с параметром, в котором введенные с клавиатуры числа через переменную **buf [буф]** записываются в файл. Запись в текстовый файл напоминает вывод информации на экран с помощью функции **cout [си аут]**.

После каждого числа в файл заносится управляющий символ **\t [слеш ти]**. При каждом обращении к текстовому файлу, а именно – при чтении из файла, считывается очередная компонента файла. В нашем примере это одно вещественное число.

Чтобы проверить содержимое созданного файла, то есть читать информацию из файла, необходимо закрыть файл.

Функция **close [клоз]** закрывает файл, открытый для записи.

Слайд 266

В фрагменте программы на слайде открывается файл для чтения.

Для того чтобы прочитать информацию из текстового файла, необходимо описать файловую переменную **ff [эф эф]** типа **ifstream [иф стрим]** для чтения из текстового файла. Открывается файл для чтения с помощью оператора **open [опэн]**. При повторном обращении к одному и тому же файлу в пределах одной программы вводится новая файловая переменная.

После открытия файла в режиме чтения из него можно считывать информацию точно так же, как и с клавиатуры, только вместо **cin [син]** нужно указать имя потока, из которого будет происходить чтение данных.

В цикле с предусловием читается каждая компонента файла и выводится на экран. Цикл продолжается до тех пор, пока не будут прочитаны все компоненты файла.

Обратите внимание на условие продолжения цикла, указанное в операторе **while [вайл]**. Цикл продолжается до тех пор, пока не будет достигнут конец файла.

Функция **eof [енд оф файл]** определяет признак конца файла.

Итак, в этой теме мы подробно рассмотрели процесс создания и обработки файлов данных, рассмотрели файлы последовательного и произвольного доступа, достоинства и недостатки работы с теми и другими типами файлов.

В зависимости от конкретной задачи выбирается принцип организации файла.

Слайд 267

Язык **C++ [си плюс плюс]** имеет ряд существенных особенностей, которые выделяют его среди других языков программирования. В значительной степени на формирование идеологии языка повлияла цель, которую ставили перед собой его создатели. Цель – обеспечение системного программиста удобным инструментальным языком, который мог бы заменить Ассемблер. В результате появился язык программирования высокого уровня, обеспечивающий необычайно легкий доступ к аппаратным средствам компьютера. Язык **C++ [си плюс плюс]** поддерживает полный набор конструкций структурного программирования, модульность, блочную структуру программы.

«Главная цель, к которой нужно стремиться, – получить легко читаемую программу возможно более простой структуры. Широко известен верный в большинстве случаев девиз – лучше по-простому, чем по-умному. Смысл девиза заключается в том, что если какое-либо действие можно запрограммировать разными способами, то предпочтение должно отдаваться не наиболее компактному и даже не наиболее эффективному, а такому, который легче для понимания. Особенно это важно в том случае, когда пишут программу одни, а сопровождают другие, что является широко распространенной практикой.

«Язык образует среду мышления и формирует представление о том, о чем мы думаем», – говорил Бенджамин Ли Уорф.

«Вы можете написать небольшую программу, используя грубую силу и нарушая все правила хорошего стиля. Для программ большего размера вы не сможете это сделать. Если структура программы, состоящей из 100 000 строк, плоха, вы обнаружите, что новые ошибки появляются с той же скоростью, с которой исправляются старые. **C++ [си плюс плюс]**

разрабатывался таким образом, чтобы предоставить возможность рационально структурировать большие программы, и чтобы один человек мог работать с большим объемом кода», – говорил Бьерн Страуструп» [5].

На этом мы завершаем изучение языка **C++ [си плюс плюс]**. Надеемся, что этот курс поможет вам в освоение дисциплин по специальности.

Желаем удачи!

Список используемых источников

1. Введение в языки программирования C и C++ | Уроки C++ – Ravesli [Электронный ресурс]. – Режим доступа: / https://ravesli.com/urok-2-vvedenie-v-yazyki-programmirovaniya-c-i-s/

2. Т.А. Павловская. С/С++ Программирование на языке высокого уровня [Электронный ресурс]. – Режим доступа: /http://cph.phys.spbu.ru/documents/First/books/7.pdf

3. Введение в программирование | Уроки C++ – Ravesli [Электронный ресурс]. – Режим доступа: /https://ravesli.com/urok-1-vvedenie-v-programmirovanie/

4. Технология программирования – Информатика, автоматизированные информационные технологии и системы [Электронный ресурс]. – Режим доступа: /https://studref.com/441961/informatika/tehnologiya_programmirovaniya

5. Бьерн Страуструп. Язык программирования C++ 11 [Электронный ресурс]. – Режим доступа: /https://vk.com/doc-145125017_450056359

6. Язык СИ++ Учебное пособие [Электронный ресурс]. – Режим доступа: / http://5fan.ru/wievjob.php?id=4301