# Объектно-ориентированное программирование на Borland C++

### Тексты лекций, прочитанных в Московском государственном университете экономики, статистики и информатики

В текстах лекций рассмотрены основные принципы и средства объектно-ориентированного программирования с применением языка С++ и системы программирования Borland C++. В приложении приведены задачи для самостоятельного решения. Предполагается, что читатель уже знаком с основами универсальных языков программирования. Пособие предназначено для студентов специальности "Прикладная математика" и может быть использовано студентами других специальностей для самостоятельного изучения С++.

## 1. Объектно-ориентированный подход в программировании

### 1.1 Технологии программирования

Технология программирования - это совокупность методов и средств разработки (написания) программ и порядок применения этих методов и средств.

На ранних этапах развития программирования, когда программы писались в виде последовательностей машинных команд, какая-либо технология программирования отсутствовала. Первые шаги в разработке технологии состояли в представлении программы в виде последовательности операторов. Написанию последовательности машинных команд предшествовало составление операторной схемы, отражающей последовательность операторов и переходы между ними. Операторный подход позволил разработать первые программы для автоматизации составления программ - так называемые составляющие программы.

С увеличением размеров программ стали выделять их обособленные части и оформлять их как подпрограммы. Часть таких подпрограмм объединялась в библиотеки, из которых подпрошраммы можно было включать в рабочие программы и затем вызывать из рабочих программ. Это положило начало процедурному программированию - большая программа представлялась совокупностью процедур-подпрограмм. Одна из подпрограмм являлась главной, и с нее начиналось выполнение программы.

В 1958 году были разработаны первые языки программирования, Фортран и Алгол-58. Программа на Фортране состояла из главной программы и некоторого количества процедур - подпрограмм и функций. Программа на Алголе-58 и его последующей версии Алголе-60 представляла собой единое целое, но имела блочную структуру, включающую главный блок и вложенные блоки подпрограмм и функций. Компиляторы для Фортрана обеспечивали раздельную трансляцию процедур и последующее их объединение в рабочую программу, первые компиляторы для Алгола предполагали, что транслируется сразу вся программа, раздельная трансляция процедур не обеспечивалась.

Процедурный подход потребовал структурирования будущей программы, разделения ее на отдельные процедуры. При разработке отдельной процедуры о других процедурах требовалось знать только их назначение и способ вызова. Появилась возможность перерабатывать отдельные процедуры, не затрагивая остальной части программы, сокращая при этом затраты труда и машинного времени на разработку и модернизацию программ.

Следующим шагом в углублении структурирования программ стало так называемое структурное программирование, при котором программа в целом и отдельные процедуры рассматривались как последовательности канонических структур: линейных участков, циклов и разветвлений. Появилась возможность читать и проверять программу как последовательный текст, что повысило производительность труда программистов при разработке и отладке программ. С целью повышения структурности программы были выдвинуты требования к большей независимости подпрограмм, подпрограммы должны связываться с вызывающими их программами только путем передачи им аргументов, использование в подпрограммах переменных, принадлежащих другим процедурам или главной программе, стало считаться нежелательным.

Процедурное и структурное программирование затронули, прежде всего, процесс описания алгоритма как последовательности шагов, ведущих от варьируемых исходных данных к искомому результату. Для решения специальных задач стали разрабатываться языки программирования, ориентированные на конкретный класс задач: на системы управления базами данных, имитационное моделирование и т.д.

При разработке трансляторов все больше внимания стало уделяться обнаружению ошибок в исходных текстах программ, обеспечивая этим сокращение затрат времени на отладку программ.

Применение программ в самых разных областях человеческой деятельности привело к необходимости повышения надежности всего программного обеспечения. Одним из направлений совершенствования языков программирования стало повышения уровня типизации данных. Теория типов данных исходит из того, что каждое используемое в программе данное принадлежит одному и только одному типу данных. Тип данного определяет множество возможных значений данного и набор операций, допустимых над этим данным. Данное конкретного типа в ряде случаев может быть преобразовано в данное другого типа, но такое преобразование должно быть явно представлено в программе. В зависимости от степени выполнения перечисленных требований можно говорить об уровне типизации того или иного языка программирования. Стремление повысить уровень типизации языка программирования привело к появлению языка Паскаль, который считается строго типизированным языком, хотя и в нем разрешены некоторые неявные преобразования типов, например, целого в вещественное. Применение строго типизированного языка при написании программы позволяет еще при трансляции исходного текста выявить многие ошибки использования данных и этим повысить надежность программы. Вместе с тем строгая типизация сковывала свободу программиста, затрудняла применение некоторых приемов преобразования данных, часто используемых в системном программировании. Практически одновременно с Паскалем был разработан язык Си, в большей степени ориентированный на системное программирование и относящийся к слабо типизированным языкам.

Все универсальные языки программирования, несмотря на различия в синтаксисе и используемых ключевых словах, реализуют одни и те же канонические структуры: операторы присваивания, циклы и разветвления. Во всех современных языках присутствуют предопределенные (базовые) типы данных (целые и вещественные арифметические типы, символьный и, возможно, строковый тип), имеется возможность использования агрегатов данных, в том числе массивов и структур (записей). Для арифметических данных разрешены обычные арифметические операции, для агрегатов данных обычно предусмотрена только операция присваивания и возможность обращения к элементам агрегата. Вместе с тем при разработке программы для решения конкретной прикладной задачи желательна возможно большая концептуальная близость текста программы к описанию задачи. Например, если решение задачи требует выполнения операций над комплексными числами или квадратными матрицами, желательно, чтобы в программе явно присутствовали операторы сложения, вычитания, умножения и деления данных типа комплексного числа, сложения, вычитания, умножения и обращения данных типа квадратной матрицы. Решение этой проблемы возможно несколькими путями:

- Построением языка программирования, содержащего как можно больше типов данных, и выбором для каждого класса задач некоторого подмножества этого языка. Такой язык иногда называют языком-оболочкой. На роль языка-оболочки претендовал язык ПЛ/1, оказавшийся настолько сложным, что так и не удалось построить его формализованное описание. Отсутствие формализованного описания, однако, не помешало широкому применению ПЛ/1 как в Западной Европе, так и в СССР.

- Построением расширяемого языка, содержащего небольшое ядро и допускающего расширение, дополняющее язык типами данных и операторами, отражающими концептуальную сущность конкретного класса задач. Такой язык называют языком-ядром. Как язык-ядро были разработаны языки Симула и Алгол-68, не получившие широкого распространения, но оказавшие большое влияние на разработку других языков программирования.

Дальнейшим развитием второго пути явился объектно-ориентированный подход к программированию, рассматриваемый в следующем параграфе.

### 1.2. Сущность объектно-ориентированного подхода к программированию

Основные идеи объектно-ориентированного подхода опираются на следующие положения:

- Программа представляет собой модель некоторого реального процесса, части реального мира.

- Модель реального мира или его части может быть описана как совокупность взаимодействующих между собой объектов.

- Объект описывается набором параметров, значения которых определяют состояние объекта, и набором операций (действий), которые может выполнять объект.

- Взаимодействие между объектами осуществляется посылкой специальных сообщений от одного объекта к другому. Сообщение, полученное объектом, может потребовать выполнения определенных действий, например, изменения состояния объекта.

- Объекты, описанные одним и тем же набором параметров и способные выполнять один и тот же набор действий представляют собой класс однотипных объектов.

С точки зрения языка программирования класс объектов можно рассматривать как тип данного, а отдельный объект - как данное этого типа. Определение программистом собственных классов объектов для конкретного набора задач должно позволить описывать отдельные задачи в терминах самого класса задач (при соответствующем выборе имен типов и имен объектов, их параметров и выполняемых действий).

Таким образом, объектно-ориентированный подход предполагает, что при разработке программы должны быть определены классы используемых в программе объектов и построены их описания, затем созданы экземпляры необходимых объектов и определено взаимодействие между ними.

Классы объектов часто удобно строить так, чтобы они образовывали иерархическую структуру. Например, класс “Студент”, описывающий абстрактного студента, может служить основой для построения классов “Студент 1 курса”, “Студент 2 курса” и т.д., которые обладают всеми свойствами студента вообще и некоторыми дополнительными свойствами, характеризующими студента конкретного курса. При разработке интерфейса с пользователем программы могут использовать объекты общего класса “Окно” и объекты классов специальных окон, например, окон информационных сообщений, окон ввода данных и т.п. В таких иерархических структурах один класс может рассматриваться как базовый для других, производных от него классов. Объект производного класса обладает всеми свойствами базового класса и некоторыми собственными свойствами, он может реагировать на те же типы сообщений от других объектов, что и объект базового класса и на сообщения, имеющие смысл только для производного класса. Обычно говорят, что объект производного класса наследует все свойства своего базового класса.

Некоторые параметры объекта могут быть локализованы внутри объекта и недоступны для прямого воздействия извне объекта. Например, во время движения объекта-автомобиля объект-водитель может воздействовать только на ограниченный набор органов управления (рулевое колесо, педали газа, сцепления и тормоза, рычаг переключения передач) и ему недоступен целый ряд параметров, характеризующих состояние двигателя и автомобиля в целом.

Очевидно, для того, чтобы продуктивно применять объектный подход для разработки программ, необходимы языки программирования, поддерживающие этот подход, т.е. позволяющие строить описание классов объектов, образовывать данные объектных типов, выполнять операции над объектами. Одним из первых таких языков стал язык SmallTalk в котором все данные являются объектами некоторых классов, а общая система классов строится как иерархическая структура на основе предопределенных базовых классов.

Опыт программирования показывает, что любой методический подход в технологии программирования не должен применяться слепо с игнорированием других подходов. Это относится и к объектно-ориентированному подходу. Существует ряд типовых проблем, для которых его полезность наиболее очевидна, к таким проблемам относятся, в частности, задачи имитационного моделирования, программирование диалогов с пользователем. Существуют и задачи, в которых применение объектного подхода ни к чему, кроме излишних затрат труда, не приведет. В связи с этим наибольшее распространение получили объектно-ориентированные языки программирования, позволяющие сочетать объектный подход с другими методологиями. В некоторых языках и системах программирования применение объектного подхода ограничивается средствами интерфейса с пользователем (например, Visual FoxPro ранних версий).

Наиболее используемыми в настоящее время объектно-ориентированными языками являются Паскаль с объектами и Си++, причем наиболее развитые средства для работы с объектами содержатся в Си++.

Практически все объектно-ориентированные языки программирования являются развивающимися языками, их стандарты регулярно уточняются и расширяются. Следствием этого развития являются неизбежные различия во входных языках компиляторов различных систем программирования. .Наиболее распространенными в настоящее время являются системы программирования Microsoft C++ , Microsoft Visual C++ и системы программирования фирмы Borland International. Дальнейший материал в данном пособии излагается применительно к системе программирования Borland C++. Это связано прежде всего наличием в этой системе программирования развитой интегрированной среды, объединяющей текстовый редактор, компилятор, редактор связей (компоновщик) и отладочные средства.

## 2. Начальные сведения о языке Си

### 2.1 Назначение Си, исторические сведения

Язык Си был разработан в 70-е годы как язык системного программирования. При этом ставилась задача получить язык, обеспечивающий реализацию идей процедурного и структурного программирования и возможность реализации специфических приемов системного программирования. Такой язык позволил бы разрабатывать сложные программы на уровне, сравнимом с программированием на Ассемблере, но существенно быстрее. Эти цели, в основном, были достигнуты. Большинство компиляторов для Си написаны на Си, операционная система UNIX <также почти полностью написана на Си. Недостатком Си оказалась низкая надежность разрабатываемых программ из-за отсутствия контроля типов. Попытка поправить дело включением в систему программирования Си отдельной программы, контролирующей неявные преобразования типов, решила эту проблему лишь частично.

На основе Си в 80-е годы был разработан язык Си++, вначале названный "Си с классами". Си++ практически включает язык Си и дополнен средствами объектно-ориентированного программирования. Рабочая версия Си++ появилась в 1983 г. С тех пор язык продолжает развиваться и опубликовано несколько версий проекта стандартов Си и Си++.

Рядом фирм, производящих программное обеспечение, разработаны компиляторы для Си и Си++. Системы программирования фирмы Borland International выделяются среди других фирм прежде всего комплексным подходом к разработке программ, выражающимся во включении в систему программирования интегрированной среды разработчика, объединяющей под общим управлением текстовый редактор для ввода исходных текстов программ, компилятор, редактор связей и набор отладочных средств. В 1989 г. этой фирмой была выпущена система Turbo C++, включавшая компилятор Си++, работающий в операционной системе DOS, с 1992 г. выпускаются системы Borland C++, содержащие компиляторы Си++ для DOS и WINDOWS, с 1997 г. поставляется версия Borland C 5.0, содержащая компиляторы Си++ для WINDOWS, причем компилятор для WINDOWS теперь позволяет разрабатывать как 16-разрядные, так и 32-разрядные варианты программ для ПЭВМ с процессорами i486 и Pentium.

Программа на Си/Си++ представляет собой один или несколько исходных файлов, которые могут транслироваться раздельно. Результаты трансляции (объектные файлы) объединяются в исполняемый файл редактором связей (компоновщиком). Обычно различают два типа исходных файлов: файлы заголовков и программные файлы. Файлы заголовков содержат описания типов данных и прототипов функций и предназначены для включения в программные файлы перед их компиляцией, их имена, как правило, имеют расширение .h, например, stdio.h. Программные файлы содержат описания функций и, возможно, глобальных переменных и констант, их имена принято записывать с расширениями .c или .cpp, например, myprog.cpp. Один и тот же файл заголовков может включаться в несколько программных файлов

Каждый файл содержит последовательность так называемых "внешних определений", описывающих типы данных, переменные, константы и функции.

В последующих параграфах этого раздела приведен обзор средств Си/Си++, не связанных с объектной ориентацией Си++.

### 2.2 Алфавит, базовые типы и описание данных.

Алфавит языка включает практически все символы, имеющиеся на стандартной клавиатуре ПЭВМ:

- латинские буквы A...Z, a...z;

- цифры 0...9;

- знаки операций и разделители:

{ } [ ] ( ) . , -> & \* + - ~ ! / % ? : ; = < > | # ^

Некоторые операции обозначаются комбинациями символов, значения символов операций в ряде случаев зависят от контекста, в котором они употреблены.

Базовые (предопределенные) типы данных объединены в две группы: данные целого типа и данные с плавающей точкой (вещественные).

Данные целого типа могут быть обычными целыми со знаком (signed) и целыми без знака (unsigned). По числу разрядов, используемых для представления данного (диапазону значений) различают обычные целые (int), короткие целые (short int) и длинные целые (long int ). Символьные данные (char) также рассматриваются как целые и могут быть со знаком и без знака.

Константы целого типа записываются как последовательности десятичных цифр, тип константы зависит от числа цифр в записи константы и может быть уточнен добавлением в конце константы букв L или l (тип long), U или u (тип unsigned) или их сочетания:

321 - константа типа int,

5326u - константа типа unsigned int,

45637778 - константа типа long int,

2746L - константа типа long int.

Целые константы могут записываться в восьмеричной системе счисления, в этом случае первой цифрой должна быть цифра 0, число может содержать только цифры 0 ... 7:

0777 - константа типа int,

0453377 - константа типа long.

Целые константы можно записывать и в шестнадцатеричной системе счисления, в этом случае запись константы начинается с символов 0x или 0X:

0x45F - константа типа int,

0xFFFFFFFF - константа типа unsigned long.

Константы типа char всегда заключаются в одиночные кавычки, значение константы задается либо знаком из используемого набора символов, либо целой константой, которой предшествует обратная косая черта: 'A', '\33', '\042', '\x1B'. Имеется также ряд специальных символов, которые могут указываться в качестве значений константы типа char:

'\n' - новая строка,

'\t' - горизонтальная табуляция,

'\v' - вертикальная табуляция,

'\r' - перевод каретки,

'\f' - перевод страницы,

'\a' - звуковой сигнал,

'\'' - одиночная кавычка (апостроф),

'\"' - двойная кавычка,

'\\' - обратная косая черта.

Вещественные числа могут быть значениями одного из трех типов: float, double, long double. Диапазон значений каждого из этих типов зависит от используемых ЭВМ и компилятора. Константы вещественных типов могут записываться в естественной или экспоненциальной формах и по умолчанию имеют тип double, например, 15.31, 1.43E-3, 2345.1e4. При необходимости тип константы можно уточнить, записав в конце суффикс f или F для типа float, суффикс l или L для типа long double.

Внешнее определение, объявляющее переменные, состоит из необязательного спецификатора класса памяти, спецификаторов типа и списка так называемых деклараторов-инициализаторов, каждый из которых объявляет идентификатор одной переменной и, возможно, значение, присваиваемое переменной при ее объявлении. Внешнее определение заканчивается точкой с запятой:

int i, j, k; // Три переменных типа int без явной инициализации

double x=1, y=2; //Две переменных типа double с начальными значениями 1 и 2

char c1='0'; // Переменная типа char, ее значение - код литеры 0

Текст, записанный в этих примерах после знаков //, является комментарием и служит только для документирования программы. Такой комментарий может занимать только одну строку текста и допускается в текстах программ на Си++. Комментарий, занимающий несколько строк, заключается в специальные скобки /\* и \*/.

В качестве спецификаторов класса памяти во внешнем определении может указываться одно из ключевых слов extern, static или typedef, Спецификатор extern означает, что объявляемый объект принадлежит другому программному файлу, а здесь дается информация о его имени и типе и не должно присутствовать инициализирующее выражение. Спецификатор static ограничивает область действия объявляемого имени данным файлом или блоком, если объявление содержится в блоке.

Если объявление данного содержится внутри тела функции (локальное объявление), то можно указывать спецификаторы класса памяти register или auto. Спецификатор register носит рекомендательный характер, компилятор пытается разместить данное этот класса в регистре процессора, если в данный момент имеются свободные регистры. Спецификатор auto принимается по умолчанию и поэтому явно не указывается, он означает, что данное класса auto должно размещаться в программном стеке при вызове функции.

Спецификатор typedef служит для присвоения имени описываемому типу данного и будет рассмотрен подробнее в следующем параграфе.

Наряду с показанными выше константами-литералами, значения которых определяются их представлением в программе, в Си и Си++ предусмотрены константы, которым присваиваются собственные имена - именованные константы. В описании именованной константы присутствует описатель const, например,

const double Pi = 3.141592653;

Переменной, идентификатор которой объявлен с описателем const, нельзя присвоить иное значение, чем было установлено при объявлении идентификатора. Инициализирующее значение при объявлении константы является обязательным.

Наряду с базовыми целыми и вещественными типами различных размеров в программе могут объявляться и использоваться данные типов, определяемых программистом: указатели, ссылки, агрегаты данных и данные перечислимого типа.

Перечислимый тип применяется для данных целого типа, которые могут принимать ограниченный набор значений. Каждому значению соответствует собственное имя-идентификатор и целое число, значение этого имени. Объявление перечислимого типа строится по схеме:

enum идентификатор {список перечисления} деклараторы-инициализаторы;

###### Здесь идентификатор задает имя перечислимого типа, список перечисления состоит из перечислителей, разделенных запятыми. Каждый перечислитель задается идентификатором и, возможно, целым значением типа char или int, например,

enum color { RED, GREEN, BLUE } en\_color;

enum lex\_type { CNST, VAR, OPER=3, FUNC };

###### Если значение перечислителя не задано, первый из них получает значение 0, а каждый следующий - значение, большее на 1. Вообще любой перечислитель по умолчанию имеет значение на 1 больше предыдущего. В Си/Си++ принято записывать идентификаторы перечислителей прописными буквами. Имена перечислителей используется либо как именованные константы либо для присвапивания переменным перечислимого типа.

В Си/Си++ для ссылок на переменную того или иного типа служат указатели. Указатель - это тип данного, значением которого является адрес другого данного. При объявлении указателя перед идентификатором записывается знак \*. Указатель может инициализироваться адресом данного, для получения адреса служит операция & (амперсенд):

double y;

double \*px, \*py = &y;

Для указателей определены операции сравнения, сложения указателя с целым числом, вычитание двух указателей, а также операция индексирования (операция []).

Для обращения к переменной по указателю выполняется операция разыменования, обозначаемая знаком \* (звездочка), например, \*py = 7.5; .

При объявлении указателя может использоваться описатель const, например,

const int cc = 20;

const int \*pc = &cc; // Можно инициализировать адресом константы.

double \*const delta = 0.001; // Указатель - константа

Кроме обычных переменных и указателей в Си++ имеется тип "ссылка на переменную", задающий для переменной дополнительное имя (псевдоним). Внутреннее представление ссылки такое же, как указателя, т.е. в виде адреса переменной, но обращение к переменной по ссылке записывается в той же форме, что и обращение по основному имени. Переменная типа ссылки всегда инициализируется заданием имени переменной, к которой относится ссылка. При объявлении ссылки за именем типа записывается знак & (амперсенд):

int ii;

int& aii = ii;

При таком описании операторы aii = 5; и ii = 5; эквивалентны.

Из переменных любого типа могут образовываться массивы. При объявлении массива в деклараторе-инициализаторе за идентификатором массива задается число элементов массива в квадратных скобках:

int a [ 5 ] ; // Массив из пяти элементов типа int

Индексы элементов массива всегда начинаются с 0, индекс последнего элемента на единицу меньше числа элементов в массиве. Массив может инициализироваться списком значений в фигурных скобках:

int b [ 4 ] = { 1, 2, 3, 4 };

При наличии списка инициализации, охватывающего все элементы массива, можно не указывать число элементов массива, оно будет определено компилятором:

int c [ ] = { 1, 2, 3 }; // Массив из трех элементов типа int

Массивы с размерностью 2 и более рассматриваются как массивы массивов и для каждого измерения указывается число элементов:

double aa [ 2 ] [ 2 ] = { 1, 2, 3, 4 }; // Матрица 2 \* 2

Массивы типа char могут инициализироваться строковым литералом. Строковый литерал - это последовательность любых символов, кроме кавычек и обратной косой черты, заключенная в кавычки. Если строковый литерал не умещается на одной строке, его можно прервать символом "\" и продолжить с начала следующей строки. В стандарте C++ предусмотрена и другая возможность записи длинных литералов в виде нескольких записанных подряд строковых литералов. Если между строковыми литералами нет символов, отличных от пробелов, такие литералы сливаются компилятором в один.

При размещении в памяти в конце строкового литерала добавляется символ '\0', т.е. нулевой байт. Строковый литерал может применяться и для инициализации указателя на тип char:

char str1 [ 11 ] = "Это строка",

str2 [ ] = " Размер этого массива определяется"

" числом знаков в литерале + 1";

char \*pstr = "Указатель с инициализацией строкой";

Имя массива в Си/Си++ является указателем-константой, ссылающимся на первый элемент массива, имеющий индекс, равный нулю. Для обращения к элементу массива указывается идентификатор массива и индекс элемента в круглых скобках, например, c[2], aa[0][1].

### 2.3 Структуры и объединения

Наряду с массивами в Си/Си++ имеются агрегаты данных типа структур и объединений. Тип структуры представляет собой упорядоченную совокупность данных различных типов, к которой можно обращаться как к единому данному. Описание структурного типа строится по схеме:

struct идентификатор

{ деклараторы членов } деклараторы\_инициализаторы;

Такое объявление выполняет две функции, во-первых объявляется структурный тип, во-вторых объявляются переменные этого типа.

Идентификатор после ключевого слова struct является именем структурного типа. Имя типа может отсутствовать, тогда тип будет безымянный и в других частях программы нельзя будет объявлять данные этого типа. Деклараторы\_инициализаторы объявляют конкретные переменные структурного типа, т.е. данные описанного типа, указатели на этот тип и массивы данных. Деклараторы\_инициализаторы могут отсутствовать, в этом случае объявление описывает только тип структуры.

Структура, описывающая точку на плоскости, может быть определена так:

struct Point\_struct // Имя структуры

{ int x, y; } // Деклараторы членов структуры

point1, \*ptr\_to\_point, arpoint [3]; // Данные структурного типа

Члены (компоненты) структуры описываются аналогично данным соответствующего типа и могут быть скалярными данными, указателями, массивами или данными другого структурного типа. Например, для описания структурного типа "прямоугольник со сторонами, параллельными осям координат" можно предложить несколько вариантов:

struct Rect1

{Point p1; // Координаты левого верхнего угла

Point p2; // Координаты правого нижнего угла

};

struct Rect2

{Point p [ 2 ];

};

struct Rect3

{Point p; // Левый верхний угол

int width; // Ширина

int high; // Высота прямоугольника

};

Поскольку при описании членов структуры должны использоваться только ранее определенные имена типов, предусмотрен вариант предварительного объявления структуры, задающий только имя структурного типа. Например, чтобы описать элемент двоичного дерева, содержащий указатели на левую и правую ветви дерева и указатель на некоторую структуру типа Value, содержащую значение данного в узле, можно поступить так:

struct Value;

struct Tree\_element

{Value \* val;

Tree\_element \*left, \*right;

};

Членами структур могут быть так называемые битовые поля, когда в поле памяти переменной целого типа (int или unsigned int) размещается несколько целых данных меньшей длины. Пусть, например, в некоторой програме синтаксического разбора описание лексемы содержит тип лексемы (до шести значений) и порядковый номер лексемы в таблице соответствующего типа (до 2000 значениий). Для представления значения типа лексемы достаточно трех двоичных разрядов (трех бит), а для представления чисел от 0 до 2000 - 11 двоичных разрядов (11 бит). Описание структуры, содержащей сведения о лексеме может выглядеть так:

struct Lexema

{unsigned int type\_lex : 3;

unsigned int num\_lex :11;

};

Двоеточие с целым числом после имени члена структуры указывает, что это битовое поле, а целое число задает размер поля в битах.

Объединение можно определить как структуру, все компоненты которой размещаются в памяти с одного и того же адреса. Таким образом, объединение в каждый момент времени содержит один из возможных вариантов значений. Для размещения объединения в памяти выделяется участок, достаточный для размещения члена объединения самого большого размера. Применение объединения также позволяет обращаться к одному и тому же полю памяти по разным именам и интерпретировать как значения разных типов.

Описание объединения строится по той же схеме, что и описание структуры, но вместо ключевого слова struct используется слово union, например, объединение uword позволяет интерпретировать поле памяти либо как unsigned int, либо как массив из двух элементов типа unsigned char.

union uword

{unsigned int u;

unsigned char b [ 2 ];

};

Описания типов, объявляемых программистом, в том числе структур и объединений могут быть достаточно большими, поэтому в Си/Си++ предусмотрена возможность присваивания типам собственных имен (синонимов), достигая при этом повышения наглядности программных текстов. Синоним имени типа вводится с ключевым словом typedef и строится как обычное объявление, но идентификаторы в деклараторах в этом случае интерпретируются как синонимы описанных типов. Синонимы имен типов принято записывать прописными буквами, чтобы отличать их от идентификаторов переменных. Ниже приведено несколько примеров объявления синонимов имен типов.

typedef struct { double re, im } COMPLEX;

typedef int \*PINT;

После таких объявлений синоним имени может использоваться как спецификатор типа:

COMPLEX ca, \*pca; // переменная типа COMPLEX и указатель на COMPLEX

PINT pi; // указатель на int

Приведенное выше описание структур и объединений в основном соответствует их построению в языке Си. В Си++ структуры и объединения являются частными случаями объектных типов данных. Дополнительные сведения об этом будут приведены при рассмотрении объектно-ориентированных средств Си++.

### 2.4 Операции и выражения

Несмотря на ограниченный набор базовых типов данных (целые и вещественные арифметические данные и строковые литералы) в языке Си++ определен обширный набор операций над данными, часть из которых непосредственно соответствует машинным командам. Как и во всех языках программирования, операции служат для построения выражений. Выражение представляет собой последовательность операндов и знаков операций и служит для вычисления некоторого значения.

В качестве операндов в выражении выступают идентификаторы переменных, константы, и строковые литералы, являющиеся первичными выражениями. Выражение, заключенное в круглые скобки, также рассматривается как первичное. Каждая операция предполагает использование определенных типов операндов (целых, вещественных, указателей). Операция присваивания в Си++ также является выражением, в связи с этим различаются операнды, которым можно присвоить новое значение и операнды, значение которых не может меняться. Чтобы операнду можно было присвоить значение, ему должна соответствовать область памяти и компилятору должен быть известен адрес этой памяти. Такие операнды называют L-выражениями (от английского left -левый), так как они могут быть записаны в левой части оператора присваивания.

Результат вычисления выражения зависит от приоритетов операций. В Си++ сложная система приоритетов операций, включающая 16 уровней. В таблице 2.1 приведен перечень операций Си++ с указанием их приоритетов, назначения и схемы записи.

### Таблица 2.1

|  |  |  |  |
| --- | --- | --- | --- |
| **Приоритет** | **Знак операции** | **Назначение** | **Схема** |
| 1 | : : | Доступ к глобальному имени или имени из другой области | : : идентификатор (глобальный) имя области : : имя\_члена\_структуры |
| 1 | -> | Обращение к члену структуры по указателю на структуру | указатель -> имя\_члена\_структуры |
| 1 | . | Обращение к члену структуры по имени структуры | имя\_структуры . имя\_члена\_структуры |
| 1 | [ ] | Обращение к элементу массива | указатель [ индекс ] |
| 1 | ( ) | Преобразование типа данного | имя\_типа (выражение ) или (тип) выражение |
| 1 | ( ) | Вызов функции | функция(аргументы) |
| 2 | ++ | Автоувеличение | ++ L-значение или L-значение++ |
| 2 | -- | Автоуменьшение | -- L-значение или L-значение-- |
| 2 | ~ | Битовое инвертирование | ~ целое\_выражение |
| 2 | ! | Логическое отрицание | ! выражение |
| 2 | - | Одноместный минус | - выражение |
| 2 | + | Одноместный плюс | + выражение |
| 2 | & | Получение адреса | & L-значение |
| 2 | \* | Разыменование указателя | \* указатель |
| 2 | new | Выделение динамической памяти | new тип данного |
| 2 | delete | Освобождение памяти | delete указатель |
| 2 | delete [] | Освобождение памяти для массива | delete [] указатель |
| 2 | sizeof | Размер данного | sizeof выражение |
| 2 |  | Размер типа данного | sizeof (имя типа ) |
| 3 | \* | Умножение | выражение \* выражение |
| 3 | / | Деление | выражение / выражение |
| 3 | % | Остаток от деления нацело | выражение % выражение |
| 4 | ->\* | Обращение к члену структуры по указателю | указатель\_на\_структуру ->\* имя\_члена\_структуры-указателя |
| 4 | .\* | Обращение к члену структуры по имени структуры | имя\_структуры .\* имя\_члена\_структуры-указателя |
| 5 | + | Сложение | выражение + выражение |
| 5 | - | Вычитание | выражение - выражение |
| 6 | << | Сдвиг влево | целое\_выражение << целое\_выражение |
| 6 | >> | Сдвиг вправо | целое\_выражение >> целое\_выражение |
| 7 | < | Меньше | выражение < выражение |
| 7 | <= | Меньше или равно | выражение <= выражение |
| 7 | > | Больше | выражение > выражение |
| 7 | >= | Больше или равно | выражение >= выражение |
| 8 | == | Равно | выражение == выражение |
| 8 | != | Не равно | выражение != выражение |
| 9 | & | Поразрядная конъюнкция | выражение & выражение |
| 10 | ^ | Отрицание равнозначности | выражение ^ выражение |
| 11 | | | Поразрядная дизъюнкция | выражение | выражение |
| 12 | && | Логическое "И" | выражение && выражение |
| 13 | | | | Логическое "ИЛИ" | выражение | | выражение |
| 14 | ? : | Условное выражение | выражение ? выражение1 : выражение2 |
| 15 | = | Простое присваивание | выражение = выражение |
| 15 | @= | Составное присваивание, знак @ - один из знаков операций \* / % + - << >> & ^ | | выражение @= выражение |
| 16 | , | Операция следования | выражение , выражение |

Рассмотрим особенности применения основных из перечисленных выше операций.

Операция ": :" (два двоеточия) применяется для уточнения имени объекта программы в случае, когда в этом месте программы известны два одинаковом имени, например, когда одно имя объявлено глобально, а другое в теле функции. Если имени предшествуют два двоеточия, то это глобальное имя.

Для обращения к членам структуры или объединения можно воспользоваться либо именем структурного данного, либо указателем на структурное данное. В первом случае полное имя члена структуры состоит из имени самой структуры и имени члена структуры, разделенных точкой. Во втором случае за именем указателя на структуру ставится знак -> (стрелка), а за ним имя члена структуры. Пусть в программе объявлен структурный тип AnyStruct, содержащий компоненту с именем member типа int и объявлены

AnyStruct s1; // Данное s1 типа AnyStruct

AnyStruct \*ps1 = &s1; // Указатель на данное типа AnyStruct

Тогда к члену структуры member из s1 можно обратиться как к s1.member или как ps1->member.

Поскольку членом структуры может быть указатель, в Си++ имеются специальные операции разыменования такого указателя, операции .\* и ->. Пусть одним из членов структуры AnyStruct является указатель pp1 на данное типа int. Тогда выражения s1.\*pp1 и ps1->\*pp1 обеспечат доступ к значению данного, на которое указывает pp1 из s1.

Выше отмечалось, что имя массива в Си/Си++ интерпретируется как указатель-константа на первый элемент массива. Для разыменования указателя, т.е. для доступа к данному по указателю на это данное служит операция \* (звездочка). Следовательно, если в программе объявлен массив

int Array1 [ 10 ];

###### то выражение \*Array1=0 служит для присвоения нулевого значения первому элементу массива. Чтобы обратиться к произвольному элементу массива, нужно указать индекс элемента, например, Array1 [3]. Это выражение эквивалентно выражению \*(Array1 + 3), т.е. требуется сначала увеличить указатель Array1 на 3 единицы, а затем разыменовать полученный указатель. При сложении указателя на объект некоторого типа T с целым числом N значение указателя увеличивается на N, умноженное на длину данного типа T. Отметим, что индекс можно задавать не только для имен массивов, но и для любого типа указателя, кроме указателя на тип void:

int \*pint = &Array[ 4 ]; pint [ 2 ] =1;

В этом примере указатель pint инициализирован адресом пятого элемента массива Array, а затем седьмому элементу этого массива присвоено значение 1.

В качестве индекса может задаваться любое выражение со значением целого типа.

Поскольку Си++ является типизированным языком, в нем определены явные и неявные преобразования типов данных. Неявные преобразования выполняются при двуместных арифметических операциях и операции присваивания и называются стандартными арифметическими преобразованиями. Эти преобразования выполняются в следующей последовательности:

- если один операнд имеет тип long double, другой операнд преобразуется в тип long double;

- иначе, если один операнд имеет тип double, другой операнд преобразуется в тип double;

- иначе, если один операнд имеет тип float, другой операнд преобразуется в тип float;

- иначе, если один операнд имеет тип unsigned long int, другой операнд преобразуется в тип unsigned long int;

- иначе, если один операнд имеет тип long int, >другой операнд преобразуется в тип long int;

- иначе, выполняются стандартные преобразования для целых, при этом типы char, short int и битовые поля типа int преобразуются в тип int, затем, если один операнд имеет больший размер (больший диапазон значений), чем другой операнд, то второй операнд преобразуется к типу операнда большего размера;

- в остальных случаях операнды имеют тип int.

Явное преобразование типов может быть задано в двух формах. Первая форма совместима с Си, в ней за именем типа в круглых скобках записывается преобразуемое значение, которое может быть первичным выражением или выражением с одноместной операцией. Имя типа в этом случае может быть представлено последовательностью описателей, например, (long int \* ) pp определеяет преобразование некоторого данного pp в тип указателя на long int. Вторая форма преобразования типа записывается как вызов функции, при этом имя типа должно задаваться идентификатором, например, int (x ). Следует отметить, что результат явного преобразования не является L-значением.

Операции автоувеличения и автоуменьшения ( ++ и -- ) могут быть префиксными и постфиксными и вызывают увеличение (уменьшение) своего операнда на единицу, т.е. выражение ++x эквивалентно x = x +1, а --x эквивалентно x = x - 1. Префиксная операция выполняется до того, как ее операнд будет использован в вычислении выражения, а постфиксная операция выполняется после того, как ее операнд будет использован в выражении, например, в результате вычисления выражения

++x \* 2 + y-- \*3

переменная x сначала увеличивается на 1, а затем умножается на 2, переменная y сначала умножается на 3, затем уменьшается на 1. Если перед вычислением этого выражения x и y были равны 1, то результат выражения будет равен 5, кроме того переменная x получит значение 2, а переменная y - значение 0. Таким образом, операции автоувеличения и автоуменьшения всегда дают побочный эффект, изменяют значения своих операндов. Операнды этих операций должны быть L-значениями.

Операция ~ (тильда) применяется только к целому значению и заменяет все биты своего операнда со значением 0 на 1, а биты со значением 1 на 0.

Логическое отрицание (операция !) возвращает значение 0 целого типа, если операнд не равен нулю, или значение 1, если операнд равен нулю.

Операции "одноместный +" и "одноместный -" имеют обычный математический смысл, знак + не изменяет значения операнда, знак - меняет знак операнда на противоположный.

Для получения адреса операнда, являющегося L-значением, применяется операция & (амперсанд). Результатом этой операции будет указатель на соответствующий тип данного. Разыменование указателя, т.е. получение значения данного по указателю на него обеспечивается операцией \* (звездочка). Результат операции разыменования является L-значением.

В Си++ определены операции размещения данных в динамической памяти и удаления динамических данных из памяти.

Операция new требует в качестве операнда имени типа и предназначена для размещения данного указанного типа в динамической памяти, результатом операции будет указатель на данное. При невозможности выделить память операция new возвращает значение NULL - предопределенную константу, имеющую нулевое значение практически во всех компиляторах Си и Си++. Память, выделяемую операцией new, можно инициализировать, указав за именем типа скалярного данного начальное значение в круглых скобках, задание начальных значений для агрегатов данных будет рассмотрено позже. Примеры применения операции new :

int \*ip = new int; /\* создание объекта типа int и получение указателя на него \*/

int \*ip2 = new int(2); // то же с установкой начального значения 2

inr \*intArray = new int [ 10 ]; // массив из 10 элементов типа int

double \*\*matr = new double [ m ] [ n ]; // матрица из m строк и n столбцов

Данное, размещенное в динамической памяти операцией new, удаляется из памяти операцией delete с операндом-указателем, значение которого получено операцией new, например,

delete intArray; delete ip2;

Операция delete только освобождает динамическую память, но не изменяет значение указателя-операнда. Программист должен помнить, что после освобождения памяти использовать этот указатель для обращения к данному нельзя.

Размер данного или типа данного в байтах можно получить по операции sizeof. Операнд может быть любого типа, кроме типа функции и битового поля. Если операндом является имя типа, оно должно заключаться в скобки. Возвращаемое значение имеет предопределенный тип size\_t, это целый тип, размер которого определяется реализацией компилятора, обычно типу size\_t соответствует unsigned int. Размер массива равен числу байт, занимаемых массивом в памяти, размер строкового литерала - это число знаков в литерале +1, т.е. завершающий нулевой байт учитывается при определении длины литерала. Значение, возвращаемое sizeof является константой.

Двуместные арифметические операции умножения ( \* ), деления ( / ), получения остатка от деления нацело ( % ), сложения ( + ) и вычитания ( - ) имеют обычный смысл и обычный относительный приоритет. Если операнды арифметической операции имеют разные типы, предварительно выполняются стандартные арифметические преобразования и тип результата операции определяется общим типом операндов после стандартных преобразований. Следовательно, выражение 7/2 будет иметь значение 3 типа int, так как оба опернда имеют тип int, а выражение 7.0/2 даст результат 3.5 типа double, поскольку этот тип имеет первый операнд.

Операции отношения двух выражений (<, <=, >, >=) требуют операндов арифметического типа или же оба операнда должны быть указателями на одинаковый тип. В случае операндов арифметического типа вычисляются значения операндов, выполняются стандартные арифметические преобразования и возвращается 1 типа int, если отношение выполняется (истинно), или 0, если отношение не выполняется (ложно). Когда сравниваются два указателя, результат зависит от относительного размещения в памяти объектов, на которые ссылаются указатели. Операции сравнения ( == и != ) выполняются аналогичным образом, но имеют меньший приоритет.

Выражения отношений могут соединяться логическими связками && (конъюнкция, логическое умножение) и | | (дизъюнкция, логическое сложение). В общем случае операндами логических связок могут быть любые скалярные значения и операция && дает реззультат, равный 1 типа int, если оба операнда имеют ненулевые значения, а операция | | дает результат, равный 0, если значения обоих операндов нулевые. Применяется сокращенная форма вычисления значения логических связок, если в операции && первый операнд равен нулю, то второй операнд не вычисляется и возвращается 0, если в операции | | первый операнд не равен нулю, то второй операнд не вычисляется и возвращается значение 1.

Как уже отмечалось, присваивание, обозначаемое знаком = в Си/Си++ рассматривается как операция и возвращает значение, которое было присвоено левому операнду. Операция присваивания вычисляется справа налево, т.е. сначала вычисляется присваиваемое значение, затем выполняется присваивание. Это позволяет записывать выражения вида x = y = z = 1 для установки одинаковых значений нескольким переменным. Можно, хотя это и снижает наглядность программы, строить и выражения с побочным эффектом вида (x = 2) \* (y = 3) + (z = 4 ). Результатом этого выражения будет 24, но одновременно переменные x, y и z получат новые значения.

Кроме простого присваивания имеется набор составных операций присваивания, в которых присваивание совмещается с указанной двуместной операцией. Запись x += y эквивалентна выражению x = x + y.

Для целых операндов определены операции сдвига влево и вправо. При выполнении операции e1 << e2 биты первого операнда сдвигаются влево на e1 разрядов и результат имеет тип первого операнда. Освобождающиеся правые разряды заполняются нулями. При сдвиге вправо ( e1 >> e2 ) если e1 имеет тип unsigned, освобождающиеся левые разряды заполняются нулями, а при e1 типа signed в освобождающихся левых разрядах повторяется знаковый разряд.

Над целыми операндами допустимы операции поразрядного логического умножения, логического сложения и исключающего или (отрицания равнозначности). В этих операциях операнды рассматриваются как последовательности битов и операция выполняется над каждой парой соответствующих разрядов из обоих операндов. Например, результатом выражения ( x >> ( p - n +1)) & ( ~(~0 << n )) будет выделение из целого беззнакового x n битов, начиная с бита с номером p, и сдвиг выделенных битов вправо, т.е. выделение n-разрядного целого, хранящегося в машинном слове x начиная с p-го разряда.

В Си/Си++ имеется конструкция, которая называется условным выражением. Условное выражение строится по схеме:

условие ? выражение1 : выражение2

В качестве условия может выступать любое скалярное выражение. Если результат вычисления условия ненулевой, то значением всего выражения будет выражение1, при нулевом значении условия значение всего выражения определяется выражением2. Второй и третий операнды условного выражения должны быть либо оба арифметического типа, либо однотипными структурами или объединениями, либо указателями одинакового типа, либо один из них - указатель на какой-либо тип, а другой операнд NULL или имеет тип void\*. Выражение x > 0 ? 1 : 0 возвращает 1, если x больше 0, и 0 в противном случае.

Выражение может быть представлено последовательностью выражений, разделенных запятыми, в этом случае вычисляются все выражения слева направо и возвращается значение последнего выражения в списке. Например в результате вычисления выражения

x = 2, e \* 3, x +1 будет получено значение 3 и попутно x получит значение 2. Результат умножения e \* 3 никак не может быть использован.

### 2.5 Операторы Си++

Операторы - это синтаксические конструкции, определяющие действия, выполняемые программой. В Си/Си++ имеются следующие типы операторов: операторы-выражения, операторы выбора, операторы цикла и оператор перехода. Синтаксис некоторых операторов содержит выражения, играющие роль условий, в зависисмости от выполнения или невыполнения которых выбирается та или иная последовательность действий. Поскольку в Си нет булевых выражений, в качестве условий используются любые выражения, дающие скалярные значения, и условие считается выполненым, если это значение отлично от нуля, и невыполненным, если оно равно нулю. Несколько операторов могут быть объединены в составной оператор заключением их в фигурные (операторные) скобки. Признаком конца оператора ( кроме составного оператора) служит точка с запятой, являющаяся в этом случае частью оператора.

Перед любым оператором может быть записана метка в виде идентификатора, отделенного от помечаемого оператора двоеточием. Метка служит только для указания ее в операторе перехода.

Наиболее простым является оператор-выражение, представляющий собой полное выражение, закнчивающееся точкой с запятой, например,

x = 3; y = (x +1) \* t; i++;

Выражение, оформленное как оператор, вычисляется, но его значение теряется и действие оператора-выражения состоит в побочных эффектах, сопровождающих вычисление, например, при выполнении операций присваивания, автоувеличения и автоуменьшения.

Операторы выбора в Си/Си++ представлены условным оператором и переключателем. Условный оператор аналогичен условным операторам других языков программирования и может использоваться в сокращенной и полной формах, которым соответствуют схемы:

if (выражение-условие) оператор

if (выражение-условие) оператор-1 else оператор-2

В сокращенной форме условного оператора вычисляется выражение-условие и, если его значение отлично от нуля, выполняется следующий за условием оператор, в противном случае не производится никаких действий.

В полной форме условного оператора при ненулевом значении выражения-условия выполняется оператор-1 с последующим переходом к следующему оператору программы, а при нулевом значении выражения условия выполняется оператор-2 с переходом к следующему оператору программы.

Переключатель позволяет выбрать одну из нескольких возможных ветвей высислений и строится по схеме:

switch (целое выражение) оператор.

Оператор в этом случае представляет собой тело переключателя, практически всегда является составным и имеет такой вид:

{case константа-1: операторы

case константа-2: операторы

default: операторы

};

Выполнение переключателя состоит в вычислении управляющего выражения и переходе к группе операторов, помеченных case-меткой, равной управляющему выражению, если такой case-метки нет, выполняются операторы по метке default. Пункт default может отсутствовать и тогда, если управляющему выражению не соответствуют ни одна case-метка, весь переключатель эквивалентен пустому оператору. Следует учитывать, что при выполнении переключателя происходит переход на оператора с выбранной case-меткой и дальше операторы выполняются в естественном порядке. Например, в переключателе

switch (count)

{case 1 : x=1;

case 2 : x=2;

case 3 : x=3;

default : x=4;

};

###### если значение count равно 1, то после перехода на case 1 будут выполнены все операторы, в результате x станет равным 4. Чтобы разделить ветви переключателя, в конце каждой ветви нужно записать оператор break, не имеющий операндов. По этому оператору происходит выход из переключателя к следующему оператору программы:

switch (count)

{case 1 : x = 1; break;

case 2 : x = 2; break;

case 3 : x = 3; break;

default : x = 4;

};

###### Теперь в зависимости от значения count будет выполняться только одна ветвь переключателя и x будет принимать одно из четырех предусмотренных значений.

В Си/Си++ имеется три варианта оператора цикла: цикл с предусловием, цикл с постусловием и цикл с параметром.

Цикл с предусловием строится по схеме

while (выражение-условие) оператор.

При каждом повторении цикла вычисляется выражение-условие и если значение этого выражения не равно нулю, выполняется оператор - тело цикла. Цикл с постусловием строится по схеме:

do оператор while (выражение-условие).

Выражение-условие вычисляется и проверяется после каждого повторения оператора - тела цикла, цикл повторяется, пока условие выполняется. Тело цикла в цикле с постусловием выполняется хотя бы один раз.

Цикл с параметром строится по схеме:

for (E1; E2; E3) оператор

где E1, E2 и E3 - выражения скалярного типа. Цикл с параметром реализуется по следующему алгоритму:

1. Вычисляется выражение E1. Обычно это выражение выполняет подготовку к началу цикла.

2. Вычисляется выражение E2 и если оно равно нулю выполняется переход к следующему оператору программы ( выход из цикла ). Если E2 не равно нулю, выполняется шаг 3.

3. Выполняется оператор - тело цикла.

4. Вычисляется выражение E3 - выполняется подготовка к повторению цикла, после чего снова выполняется шаг 2.

Пусть требуется подсчитать сумму элементов некоторого массива из n элементов.

С использованием цикла с предусловием это можно сделать так:

int s=0;

int i=0;

while (i < n) s +=a[ i ++];

Эта же задача с применением цикла с постусловием решается следующими операторами:

int s = 0;

int i = 0;

do s +=a[ i++]; while ( i < n );

Поскольку в данном случае повторениями цикла управляет параметр i, эту задачу можно решить и с помощью цикла третьего типа:

int i, s;

for ( s = 0, i = 0; i < n; i++) s +=a[ i ];

Объявления переменных можно внести в выражение E1 оператора for и все записать в виде одного оператора for с пустым телом цикла:

for ( int i = 0, s = 0; i < n; s += a [i++] ) ;

Для прерывания повторений оператора цикла любого типа в теле цикла может быть использован оператор break. Для перехода к следующему повторению цикла из любого места тела цикла может быть применен оператор continue. Эти операторы по своему назаначению аналогичны соответствующим операторам языка Паскаль.

Несмотря на то, что Си++ содержит полный набор операторов для структурного программирования, в него все же включен оператор перехода:

goto метка;

Метка задает адрес перехода и должна помечать оператор в том же составном операторе, которому принадлежит оператор goto. Вход в составной оператор, содержащий объявления данных, не через его начало, запрещен.

### 2.6 Функции

Любая программа на Си/Си++ содержит хотя бы одну функцию. Алгоритм решения любой задачи реализуется путем вызовов функций. Одна из функций считается главной и имеет фиксированное имя, эта функция вызывается операционной системой при запуске программы, а из нее могут вызываться другие функции. При работе в MS DOS главная функция имеет имя main.

Описание функции имеет общий синтаксис внешнего определения и отличается от описания переменного синтаксисом декларатора-инициализатора, который содержит идентификатор (имя ) функции и список параметров в круглых скобках. Тип функции - это тип значения, возвращаемого функцией. Функция может возвращать значение любого типа, кроме массива и функции, но допускается возвращать указатель на массив или указатель на функцию. Если функция не возвращает никакого значения, тип функции обозначается ключевым словом void.

В списке параметров указываются типы и имена параметров, передаваемых в функцию при ее вызове. Если функция не требует передачи ей аргументов, список параметров может быть пустым. Для совместимости с программами, написанными на Си, рекомендуется в качестве пустого списка параметров указывать ключевое слово void.

Полное описание функции содержит также тело функции, представляющее собой составной оператор. Составной оператор - это последовательность описаний внутренних данных и операторов, заключенная в фигурные скобки. Следует отметить, что в Си/Си++ описание функций не может быть вложенным, в теле функции нельзя объявить другую функцию.

Функция, возвращающая среднее арифметическое трех вещественных данных, может быть описана так:

double sred ( double x, double y, double z)

{ double s;

s = x + y + z;

return s / 3;

};

Для вызова такой функции при условии, что предварительно объявлены переменные p, a, b, и c, можно записать оператор:

p = sred (a, b, c );

Очевидно, что функция должна быть описана до того, как встретится ее первый вызов. Это не всегда возможно, например, когда две функции вызывают друг друга. Когда программа состоит из нескольких файлов, полное описание функции должно присутствовать только в одном файле, но вызовы функции возможны из разных файлов программы. Чтобы разрешить подобные противоречия предусмотрены две формы описания функций, полная форма, называемая также определением функции, и сокращенная, называемая описанием прототипа функции или просто прототипом. Прототип функции содержит только заголовок функции и задает таким образом имя функции, тип возвращаемого значения и типы параметров. По этой информации при компиляции программного файла можно проверить правильность записи вызова функции и использования возвращаемого значения. В Си++ требуется, чтобы вызову любой функции предшествовало в том же файле либо полное определение функции либо описание ее прототипа.

Следует отметить ряд дополнительных возможностей описания функций в Си++:

- При описании прототипа функции можно не указывать имена параметров, достаточно указать их типы.

- Для части параметров функции можно задавать значение по умолчанию, что позволяет вызывать функцию с меньшим числом аргументов, чем предусмотрено описанием функции. Значение по умолчанию можно указывать только для последних параметров в списке. Например, функция sred могла бы быть описана так:

double sred ( double x, double y, double z = 0)

{ double s;

s = x + y + z;

return s / 3;

};

К такой функции можно обращаться с двумя и с тремя аргументами.

- Функция может иметь переменное число параметров, для части параметров могут быть неизвестны их типы. Неизвестная часть списка параметров обозначается многоточием. Например, функция с прототипом

int varfunc ( int n, ... );

###### имеет один обязательный параметр типа int и неопределенное число параметров неизвестных типов. Правильная интерпретация такого списка параметров в теле функции требует от программиста дополнительных усилий.

При вызове функции аргументы передаются в функцию по значениям. Это значит, что для каждого аргумента в памяти создается его копия, которая и используется при вычислении функции. Следовательно, любые изменения значений аргументов в теле функции теряются при выходе из функции. Если аргумент является массивом, в функцию передается указатель на начальный элемент массива и присваивания элементам массива в теле функции изменяют значения элементов массива аргумента.

В С++ с объявлением функции связывается так называемая сигнатура функции, определяемая типом возвращаемого значения и типами параметров. Это позволяет назначать функциям, имеющим аналогичное назначение, но использующим параметры разных типов, одинаковые имена. Например, наряду с приведенной выше функцией sred для вущественных аргументов, в той же программе может присутствовать функция

double sred ( int x, int y, int z = 0)

{int s;

s = x + y + z;

return s / 3;

};

Функция, в теле которой отсутствуют операторы цикла и переключатели, может быть объявлена с дополнительным описателем inline. В точке вызова такой функции при компиляции просто вставляется тело функции с соответствующей заменой параметром на аргументы вызова. В результате экономится время на передачу параметров, переход на подпрограмму и организацию возврата в вызывающую программу. Функции с описателем inline называют встроенными, они реализуются как открытые продпрограммы. Типичный пример такой функции - определение наибольшего (наименьшего) из двух чисел:

inline int max ( int x1, int x2)

{return x1 > x2 ? x1 : x2 }

В системах программирования Turbo C++ и Borland C++ главная функция (функция main) может принимать три параметра, что позволяет вызывать Си-программы из командной строки DOS с передачей в программу необходимых аргументов. Стандартный прототип функции main имеет вид:

int main ( int argc, char \*argv[ ], char \*envp[ ] )

В конкретной программе можно объявлять функцию main без возвращаемого значения ( возвращающую тип void ), с использованием только двух первых параметров или вообще без параметров. Параметры argc и argv служат для передачи в программу аргументов в виде массива строк, argc содержит число элементов этого массива, а argv - это массив указателей на элементы массива, причем первый элемент массива, на который указывает argv [0], содержит имя программы (имя exe-файла программы), остальные элементы представляют собой аргументы из командной строки DOS. Параметр envp используется для доступа к элементам текущей среды DOS.

### 2.7 Библиотека времени выполнения

В определении языков Си и Си++ отсутствуют операторы ввода-вывода, операции над строковыми данными и многие другие средства, имеющиеся в других языках программирования. Этот недостаток компенсируется добавлением в системы программирования Си и Си++ библиотек функций, подключаемых к рабочим программам при редактировании связей и называемых библиотеками времени выполнения. Отделение этих библиотек от компилятора позволяет в необходимых случаях использовать различные варианты этих библиотек, например, для различных моделей ЭВМ или операционных систем.

Часть функций из библиотек времени выполнения стандартизована, в стандарте зафиксированы имя функции, ее назначение, перечень и типы параметров и тип возвращаемого значения. Другие функции ориентированы на конкретные модели ЭВМ и операционные системы, способы их вызова и использования могут быть различными в системах программирования разных фирм.

Чтобы обеспечить возможность контроля правильности вызова функций при компиляции программы, в систему программирования входят файлы заголовков функций времени выполнения. В файлах заголовков определяются прототипы библиотечных функций, а так же константы и типы данных, используемые этими функциями.

Ниже приведены имена некоторых файлов заголовков и назначение описанных в них прототипов групп функций.

Стандартом Си определены следующие файлы заголовков:

ASSERT.H - Содержит макросы для сообщений об ошибках при выполнении условия, задаваемого программистом.

CTYPE.H - Функции для проверки и преобразования данных типа char.

FLOAT.H - Макросы для операций над числами с плавающей точкой.

LIMITS.H - Макросы, задающие диапазоны представления целых.

LOCALE.H - Представление даты, времени, денежных единиц.

MATH.H - Пакет стандартных математических функций.

SETJUMP.H - Имена типов и функции для реализации операторов перехода, используется редко.

SIGNAL.H - Макросы для сигнализации об ошибках согласно стандарта ANSI.

STDARG.H - Макросы для вызова функций с переменным числом аргументов.

STDDEF.H - Определение общих типов для указателей, типов size\_t и NULL.

STDIO.H - Стандартные функции ввода-вывода.

STDLIB.H - Определение общих типов, переменных и функций.

STRING.H - Функции для операций над строковыми данными.

TIME.H - Структуры и функции для операций с датами и временем.

В Си++ добавлены операции с комплексными числами и десятичными данными:

BCD.H - Данные, представленные в десятичной системе счисления

COMPLEX.H - Функции и операции над комплексными числами.

Имеются также файлы прототипов функций для распределения и освобождения динамической памяти, использования средств DOS и BIOS.

Чтобы использовать какие-либо функции из библиотек времени выполнения в программу должен быть включен файл-заголовок с прототипами требуемых функций. Включение в программу файла-заголовка обеспечивается препроцессорной директивой

# include < имя файла >

Например, для включения заголовка с функциями ввод-вывода в стиле Си в начале программы записывается строка

< PRE>

Очевидно, любая программа использует какие-либо входные данные и куда-либо выводит полученные результаты, поэтому файл заголовков stdio.h присутствует почто во всех программах. Отметим некоторые фугкции из этого файла. Функции ввода-вывода используют понятие потока, рассматриваемого как последовательность байтов. Поток может быть связан с дисковым файлом или другим устройством ввода-вывода, в том числе с консолью, когда ввод осуществляется с клавиатуры, а вывод - на экран монитора. Предусмотрены несколько стандартных потоков:

stdin - стандартный ввод,

stdout - стандартный вывод,

stderr - для вывода сообщений об ошибках,

stdprn - стандартное устройство печати,

stdaux - стандартный последовательный порт.

Потоки stdin, stdout и stderr обычно связываются с консолью, но могут быть переназначены на другие устройства. Назначение двух последних потоков зависит от используемой аппаратуры. Стандартные потоки автоматически открываются при запуске Си-программы и закрываются при ее завершении. Потоки, создаваемые программистом, открываются функцией fopen и закрываются функцией fclose.

Функции ввода-вывода из stdio.h условно можно разбить на четыре группы: ввод-вывод байтов, ввод-вывод строк, форматный ввод-вывод и так называемый прямой (бесформатный) ввод-вывод. Здесь отметим только отдельных представителей первых трех групп, предназначенных для ввода из потока stdin и вывода в поток stdout.

Функция int getchar( ) служит для ввода одного символа с клавиатуры и возвращает код символа, преобразованный к типу int. Функция int putchar (int c) выводит символ c в очередную позицию на экране монитора.

Для ввода строки с клавиатуры служит функци char \* gets ( char \* buf ), которая читает ввод с клавиатуры (до символа новой строки или нажатия клавиши Enter) и помещает коды прочитанных символов в буфер, адрес которого задается параметром buf, в конце строки добавляется нулевой байт. Вывод строки выполняет функци int puts ( char \* string), которая выводит строку по адресу string на экран, пока в строке не встретится нулевой байт и возвращает код последнего выведенного символа.

Функции форматного ввода-вывода принимают в качестве параметров строку с описанием формата представления данных на внешнем устройстве и список вводимых или выводимых данных. Строка описания формата состоит из обычных символов, управляющих символов типа новой строки, возврата каретки и т.п. и спецификаций полей ввода или вывода. Каждая такая спецификация начинается символом % (процент) за которым следуют коды флагов, размер поля ввода или вывода, число цифр в дробной части числа, префикса размера данного и кода типа формата. Обязательно указывать только тип формата, остальные компоненты спецификации формата задаются по необходимости. Отметим некоторые коды типа формата:

d - для представления целого со знаком в десятичной системе счисления,

i - для представления целого без знака в десятичной системе счисления,

f - для представления числа с плавающей точкой в естественной форме,

e или E - представление числа с плавающей точкой в экспоненциальной форме,

s - ввод-вывод строковых данных,

c - ввод-вывод символов.

Для форматного вывода служит функция int printf ( char \*format, ... ), имеющая список параметров переменной длины, количество дополнительных параметров должно соответствовать числу спецификаций формата в форматной строке, данные для вывода могут задаваться выражениями, ответственность за правильное задание спецификации формата для каждого выводимого данного полностью лежит на программисте. Пример применения функции printf :

printf ( "\n x = %d, y = %f %s", x, y, st);

При выполнении этой функции просматривается строка формата слева направо и символы не являющиеся спецификациями формата копируются на выводное устройство, управляющий символ \n переводит курсор на экране к началу следующей строки, когда встречается спецификация формата, выбирается очередной аргумент из списка вывода, преобразуется в в соответствии с форматом и выводится в очередную позицию экрана. Для правильного вывода требуется, чтобы переменная x была типа int, y - типа float, а переменная st - типа char\*.

Форматный ввод выполняет функция int scanf ( char \*format, ...) в которой список ввода должен задавать указатели на вводимые переменные. Если в строке формата присутствуют символы, не входящие в спецификации форматов и не являющиеся пробельными, то во входном потоке должны присутствовать те же символы. Пробельными символами считаются знаки пробела, табуляции, новой строки, они считываются из потока ввода, но не участвуют в формировании входных данных. Когда в форматной строке встречается спецификация формата, во входном потоке проаускаются пробельные символы, а последующие символы интерпретируются в соответствии с типом формата, преобразуются во внутреннее представление и полученное значение записывается в память по адресу очередного элемента списка ввода. Например, для ввода двух переменных целого типа и одной вещественного типа можно применить оператора вызова функции

scanf ( "%d %d %f", &x1, &x2, &y );

Здесь x1 и x2 должны быть типа int, а y - типа float. Во входном потоке вводимые значения должны разделяться хотя бы одним пробелом.

Более полную информацию о функциях ввода-вывода в стиле Си можно получить в справочной системе интегрированной среды Borland C++.

Недостатком рассмотренных выше функций ввода-вывода является отсутствие контроля соответствия типов форматов и типов вводимых (выводимых) данных, часто приводящее к ошибкам в программах. В Си++ включены собственные средства потокового ввода-вывода, обеспечивающие жесткий контроль типов в операциях ввода-вывода. Для этого определены четыре новых стандартных потока:

cin - для ввода данных,

cout - для вывода данных,

cerr - вывод сообщений об ошибках без буферизации вывода,

clog - вывод сообщений об ошибках с буферизацией вывода.

В качестве знака операции вывода определены знаки <<, а знаком операции ввода - знаки >>, те же, что и для операций сдвига. Компилятор по контексту определяет, какую операцию задают эти символы, ввод-вывод или сдвиг.

Чтобы использовать средства ввода-вывода Си++ в программу должен быть включен файл-заголовок iostream.h :

# include <iostream.h>

В операциях вывода левым операндом должен быть поток вывода, правым операндом - выводимое данное. Результатом операции вывода является поток вывода, что позволяет записывать вывод в виде цепочки операций <<, например,

cout << "x1 = " << x1 << " x2 = " << x2 << "\n";

Для базовых типов данных определены их преобразования при выводе и точность представления на устройстве вывода. Эти характеристики можно менять, применяя специальные функции, называемые манипуляторами.

В операции ввода левым операндом должен быть поток ввода, а правым операндом - имя вводимого данного для арифметических данных или указатель типа char\* для ввода строк, например,

cin >> x1 >> x2 >>st;

Операции ввода-вывода выполняются слева направо и последний оператор эквивалентен оператору ((cin>> x1) >> x2) >> st; или трем операторам

cin >> x1; cin >> x2; cin >> st;

В заключение приведем пример простой программы, запрашивающей у пользователя два целых числа и выводящей на экран их сумму:

# include <iostream.h>

int x, y ;

int main ( )

{cout << "x = "; cin >> x; // Запрос и ввод значения x

cout << "\n y = "; cin >> y; // Запрос и ввод значения y

cout << "\n" << " x + y = " << x + y;

return 0;

}.