ŋqdpγeʏ bⱻ dₑpⱻbγзe d ɘɔɑbɔʏ ɟqdpγɑɣè

МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ «ВОРОНЕЖСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Язык С++ и основы технологии объектно-ориентированного программирования

Часть I

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

По специальности ПРИКЛАДНАЯ МАТЕМАТИКА И ИНФОРМАТИКА
(01.03.02)

ВОРОНЕЖ

2017

ŋqdpγeʏ bⱻ dₑpⱻbγзe d ɘɔɑbɔʏ ɟqdpγɑɣè

1. Объектно-ориентированный подход в программировании

1.1 Технологии программирования

Технология программирования – это совокупность методов и средств разработки программ, а также 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. Этому предшествовало составление операторной схемы, отражающей последовательность выполнения операторов и осуществление переходов между ними. Операторный подход позволил разработать первые алгоритмы для автоматизации составления программ, в результате чего появились так называемые *составляющие* программы.

С увеличением размеров программ разработчики стали выделять их обособленные части и оформлять их в виде подпрограмм. Таким образом было положено начало *процедурному программированию* – большая программа представлялась совокупностью процедур-подпрограмм, одна из подпрограмм являлась главной, и с нее начиналось выполнение программы. Со временем из наборов подпрограмм стали формироваться библиотеки.

В 1958 году были разработаны первые языки программирования, Фортран (Fortran) и Алгол-58 (Algol-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 ее процедуры рассматривались как последовательности канонических структур: линейных участков, циклов и разветвлений. Появилась возможность читать и проверять программу как последовательный текст, что

- 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 переменную 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, а также 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), но ему недоступен целый ряд параметров, характеризующих состояние двигателя и автомобиля в целом.

ŋqdqɐeɎ bɆ dədɆbɎɜə d ɕɔɑbɔɐ Ɉqddɏqɐè

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 методологический 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.

Иначе говоря, объектно-ориентированный подход к программированию некоторой задачи, прежде всего, состоит в том, чтобы создать некоторый инструментарий, присущий решаемой задаче, а затем уже программировать в терминах этой задачи.

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. Дальнейший материал в данном пособии излагается на языке программирования С++. При этом особенностям его реализации в той или иной системе программирования уделено минимальное внимание.

2. Начальные сведения о языке С++

2.1 Назначение языка С++, исторические сведения

Язык Си (С) 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 С написаны на самом языке С. Операционная система UNIX также почти полностью написана на С. Недостатком С является низкая надежность разрабатываемых программ из-за отсутствия контроля типов. Попытка поправить дело вклю-

ŋqdqɐeɎ bɆ dədɆbɎɜə d ɕɔɑbɔɐ Ɉqddɏqɐè

чением в систему программирования C отдельной программы, контролирующей неявные преобразования типов, решила эту проблему лишь частично.

Язык программирования C++ был разработан сотрудником исследовательской лаборатории компании AT&T Бьерном Страуструпом (Bjarne Stroustrup) в 1980 году и изначально представлял собой попытку решения задач объектно-ориентированного программирования средствами языка C. В своих исторических замечаниях Страуструп поясняет, почему в качестве базового языка был выбран C:

- многоцелевой, лаконичный и относительно низкого уровня;
- отвечает большинству задач системного программирования;
- «идет везде и на всем»;
- пригоден в среде программирования UNIX.

Первоначальное название языка «C с классами» в 1983 году по предложению Рика Масситти (Rick Mascitti) было изменено на C++. В этом же году C++ был впервые применен за пределами исследовательской группы. С 1980 года C++ претерпел два существенных изменения: в 1985 и 1990 годах. Первый рабочий проект языка C++ стандарта ANSI (American National Standarts Institute) был представлен в январе 1994 года. 14 ноября 1997 года Международная организация стандартизации (International Standarts Organization, ISO) утвердила стандарт C++. Бьерн Страуструп высоко оценил новый стандарт, отметив, что описанная в нем реализация гораздо ближе к идеалу, чем первоначальная версия языка. Ожидалось, что фирмы-разработчики приведут свои продукты в соответствии со стандартом, что должно было радикально улучшить переносимость написанных на C++ программ.

Программа на C/C++ 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 C/C++, не связанных с объектной ориентацией C++.

2.2 Алфавит, базовые типы и описание данных

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

- 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.

Современное понятие типа базируется на множестве значений, которые могут принимать переменные данного типа, а также наборе операций, которые можно к ним применять.

Базовые (предопределенные) типы данных в языке C++ 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.

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` (`long`), `U` или `u` (`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 int`

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 int`.

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 как в формес фиксированной точкой, так и в экспоненциальной форме. По умолчанию они 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 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 про-