Тема 1.1 Стадии разработки программного обеспечения

Разработка любой программы, будь то небольшая процедура по обработке поступающей на консоль информации или комплексный программный продукт, состоит из нескольких этапов, грамотная реализация которых является обязательным условием для получения хорошего результата. Четкое следование выверенным временем этапам разработки программного обеспечения становится основополагающим критерием для занимающихся созданием ПО компаний и их заказчиков, заинтересованных в получении превосходно выполняющей свои функции программы. Подробно рассмотрим каждую стадию общепризнанной методологии разработки ПО, чтобы оценить их высокую значимость для достижения поставленной перед исполнителями цели.

Анализ требований

Самым первым этапом разработки программного обеспечения по праву называется процедура проведения всестороннего анализа выдвинутых заказчиком требований к создаваемому ПО, чтобы определить ключевые цели и задачи конечного продукта. В рамках этой стадии происходит максимально эффективное взаимодействие нуждающегося в программном решении клиента и сотрудников компании-разработчика, в ходе обсуждения деталей проекта помогающих более четко сформулировать предъявляемые к ПО требования. Результатом проведенного анализа становится формирование основного регламента, на который будет опираться исполнитель в своей работе — технического задания на разработку программного обеспечения. ТЗ должно полностью описывать поставленные перед разработчиком задачи и охарактеризовать конечную цель проекта в понимании заказчика.

Проектирование

Следующий ключевой этап в разработке программного обеспечения — стадия проектирования, то есть моделирования теоретической основы будущего продукта. Самые современные средства программирования позволяют частично объединить этапы проектирования и кодирования, то есть технической реализации проекта, будучи основанными на объектно-ориентированном подходе, но полноценное планирование требует более тщательного и скрупулезного моделирования. Качественный анализ перспектив и возможностей создаваемого продукта станет основой для его полноценного функционирования и выполнения всего комплекса возлагаемых на ПО задач. Одной из составных частей этапа проектирования, к примеру, является выбор инструментальных средств и операционной системы, которых сегодня на рынке присутствует очень большое количество.

В рамках данного этапа стороны должны осуществить:

- оценку результатов проведенного первоначально анализа и выявленных ограничений;
- поиск критических участков проекта;
- формирование окончательной архитектуры создаваемой системы;
- анализ необходимости использования программных модулей или готовых решений сторонних разработчиков;
- проектирование основных элементов продукта — модели базы данных, процессов и кода;
- выбор среды программирование и инструментов разработки, утверждение интерфейса программы, включая элементы графического отображения данных;
- определение основных требований к безопасности разрабатываемого ПО.

Кодирование

Следующим шагом становится непосредственная работа с кодом, опираясь на выбранный в процессе подготовки язык программирования. Описывать особенности и тонкости самого трудоемкого и сложного этапа вряд ли стоит, достаточно указать, что успех реализации любого проекта напрямую зависит от качества предварительного анализа и оценки конкурирующих решений, с которыми создаваемой программе предстоит «бороться» за право называться лучшей в своей нише. Если речь идет о написании кода для выполнения узкоспециализированных задач в рамках конкретного предприятия, то от грамотного подхода к этапу кодирования зависит эффективность работы компании, заказавшей разработку. Кодирование может происходить параллельно со следующим этапом разработки — тестированием программного обеспечения, что помогает вносить изменения непосредственно по ходу написания кода. Уровень и эффективность взаимодействия всех элементов, задействованных для выполнения сформулированных задач компанией-разработчиком, на текущем этапе является самым важным — от слаженности действий программистов, тестировщиков и проектировщиков зависит качество реализации проекта.

Тестирование и отладка

После достижения задуманного программистами в написанном коде следуют не менее важные этапы разработки программного обеспечения, зачастую объединяемые в одну фазу — тестирование продукта и последующая отладка, позволяющая ликвидировать огрехи программирования и добиться конечной цели — полнофункциональной работы разработанной программы. Процесс тестирования позволяет смоделировать ситуации, при которых программный продукт перестает функционировать. Отдел отладки затем локализует и исправляет обнаруженные ошибки кода, «вылизывая» его до практически идеального состояния. Эти два этапа занимают не меньше 30% затрачиваемого на весь проект времени, так как от их качественного исполнения зависит судьба созданного силами программистов программного обеспечения. Нередко функции тестировщика и отладчика исполняет один отдел, однако самым оптимальным будет распределить эти обязанности между разными исполнителями, что позволит увеличить эффективность поиска имеющихся в программном коде ошибок.

Внедрение

Процедура внедрения программного обеспечения в эксплуатацию является завершающей стадией разработки и нередко происходит совместно с отладкой системы. Как правило, ввод в эксплуатацию ПО осуществляется в три этапа:

- первоначальная загрузка данных;
- постепенное накопление информации;
- вывод созданного ПО на проектную мощность.

Ключевой целью поэтапного внедрения разработанной программы становится постепенное выявление не обнаруженных ранее ошибок и недочетов кода. В рамках этого этапа разработки программного обеспечения и заказчик, и исполнитель могут столкнуться с рядом достаточно узкого спектра ошибок, связанных с частичной рассогласованностью данных при их загрузке в БД, а также срывов выполнения программных процедур в связи с применением многопользовательского доступа. Именно на этой стадии выкристаллизовывается окончательная картина взаимодействия пользователя с программой, а также определяется степень лояльности последнего к разработанному

интерфейсу. Если выход системы на проектную мощность после ряда проведенных доработок и улучшений произошел без особых осложнений, значит предварительная работа над проектом и реализация предыдущих стадий разработки осуществлялась правильно.

Заключение

Создание даже небольшого и технически простого ПО зависит от четкого выполнения каждой фазы, то есть деятельности всех отделов, задействованных в процессе разработки. Четкий план выполнения необходимых мероприятий с указанием конечных целей становится неотъемлемой частью работы разработчиков, планирующих оставаться широко востребованными на рынке труда специалистами. Только правильно составленное техническое задание позволит добиться нужного результата и осуществить разработку по-настоящему качественного и конкурентного ПО для любой платформы — серверной, стационарной или мобильной.

Неотъемлемой частью завершающего этапа разработки программного обеспечения также является последующая техническая поддержка созданного продукта в процессе его эксплуатации на предприятии заказчика. Грамотно организованная служба техподдержки зачастую становится ключевым фактором при выборе исполнителя в рамках достижения поставленной цели.

ТЕМА 1.1. ПОНЯТИЕ АЛГОРИТМА И СПОСОБЫ ЕГО ОПИСАНИЯ

Этапы решения задачи

Можно выделить следующие основные этапы решения задачи:
• постановка (формулировка) задачи;
• построение модели, выбор метода решения задачи;
• разработка алгоритма;
• проверка правильности алгоритма;
• реализация алгоритма;
• анализ алгоритма и его сложности;
• отладка программы, обнаружение, локализация и устранение возможных ошибок;
• получение результата;
• составление документации.

Понятие алгоритма занимает центральное место в вычислительной математике и программировании. Справедливо следующее определение.

Алгоритм – строгая и четкая система правил, определяющая последовательность действий над некоторыми объектами и после конечного числа шагов приводящая к достижению поставленной цели.

Прежде чем понять задачу, необходимо ее четко сформулировать. Это условие не является достаточным для понимания задачи, но оно абсолютно необходимо.

Следующий важный шаг в решении задачи – формулировка для нее математической модели. Выбор модели и реализация алгоритма в значительной степени может повлиять на эффективность алгоритма решения задачи. Все перечисленные выше этапы нельзя рассматривать независимо друг от друга. В особенности первые три сильно влияют на последующие.

Наиболее распространенная процедура доказательства правильности программы (следующий этап) – это прогон ее на ранних тестах. Однако это не исключает все сомнения. Необходимо доказательство конечности алгоритма, при котором будут проверены все подходящие входные данные и получены все подходящие выходные данные.

Реализация алгоритма – процесс корректного преобразования алгоритма в машинную программу. Требует также построения целой системы структур данных для представления модели.

Задача анализа алгоритма и его сложности – получение оценок или границ для объема памяти или времени работы алгоритма. Полный анализ способен выявить узкие места в программах. Критерии оценок алгоритмов будут рассмотрены далее.

Эксплуатации программы предшествует отладка – исправление синтаксических и логических ошибок. Процесс проверки программы включает экспериментальное подтверждение того факта, что программа делает именно то, что должна делать. Обычно множество входов огромно, и полная проверка невозможна. Необходимо выбрать множество вводов, которые проверяют каждый участок программы. Программы следует тестировать также для того, чтобы определить их вычислительные ограничения.

Этап документации не является последним шагом в процессе построения алгоритма. Он должен переплетаться со всем процессом построения алгоритма для того, чтобы была возможность понять программы, написанные другими.

Последние этапы обеспечивают обратную связь, которая может заставить пересмотреть некоторые из предшествующих этапов.

Свойства алгоритма

Характерными свойствами алгоритма являются определенность, массовость и результативность.

Определенность (детерминированность) алгоритма предполагает такое составление предписания, которое не оставляет места для различных толкований или искажений результата, т.е. последовательность действий алгоритма строго и точно определена.

Массовость определяет возможность использования любых исходных данных из некоторого допустимого множества. Правило, сформулированное только для данного случая, не является алгоритмом (например, таблица умножения не является алгоритмом, а правило умножения «столбиком» есть алгоритм).

Результативность (конечность) алгоритма означает, что при любом допустимом исходном наборе данных алгоритм закончит свою работу за конечное число шагов.

Классификация алгоритмов

По типу используемого вычислительного процесса различают линейные (прямые), разветвляющиеся и циклические алгоритмы.

Линейные алгоритмы описывают линейный вычислительный процесс, этапы которого выполняются однократно и последовательно один за другим.

Разветвляющийся алгоритм описывает вычислительный процесс, реализация которого происходит по одному из нескольких заранее предусмотренных направлений. Направления, по которым может следовать вычислительный процесс, называются ветвями. Выбор конкретной ветви вычисления зависит от результатов проверки выполнения некоторого логического условия. Результатами проверки являются: «истина» (да), если условие выполняется, и «ложь» (нет), при невыполнении условия.

Циклический алгоритм описывает вычислительный процесс, этапы которого повторяются многократно.

Различают **простые** циклы, не содержащие внутри себя других циклов, и **сложные** (вложенные), содержащие несколько циклов.

В зависимости от местоположения условия выполнения цикла различают циклы с **предусловием** и циклы с **постусловием**.

В соответствии с видом условия выполнения циклы делятся на **циклы с параметром** и **итерационные** циклы.

Многие из реально существующих алгоритмов имеют смешанный характер, т.е. могут содержать линейные участки, разветвления, циклы с известным количеством повторений и итерационные циклы. В связи с этим составление алгоритмов сразу в законченной форме затруднено. Поэтому для составления сложных алгоритмов рекомендуется использовать нисходящее **проектирование программ (метод пошаговой детализации, метод последовательных уточнений)**. *Его суть: первоначально продумывается общая структура алгоритма, без детальной проработки его отдельных частей. Далее прорабатываются отдельные блоки, не детализированные на предыдущем шаге. Таким образом, на каждом шаге разработки уточняется реализация фрагмента алгоритма, т.е. решается более простая задача.*

Способы описания алгоритмов

Существуют следующие способы описания алгоритмов:

1) запись на естественном языке (словесное описание);

2) изображение в виде схемы (графическое описание);

3) запись на алгоритмическом языке (составление программы).

Способы словесного описания алгоритмов отличаются применяемыми метаязыками (языки, предназначенные для описания языка программирования). **Например**, словесное описание алгоритма решения квадратного уравнения $ax^2 + bx + c = 0$ будет выглядеть следующим образом:

1) $D := b^2 – 4ac$;

2) если $D < 0$, идти к 4;

3) $x_1 := (–b + \sqrt{D}) / (2a)$; $x_2 := (–b – \sqrt{D}) / (2a)$;

4) Останов.

Основной недостаток словесного описания – плохая наглядность.

Графическое описание алгоритма – это представление алгоритма в виде схемы (двухмерного рисунка), состоящей из последовательности блоков (геометрических фигур), каждый из которых отображает содержание очередного шага алгоритма (управляющей структуры). Внутри фигур кратко записывают выполняемое действие. Такую схему называют **блок-схемой алгоритма**.

На рисунке 1 приведены основные условные обозначения, используемые при графической записи алгоритма.

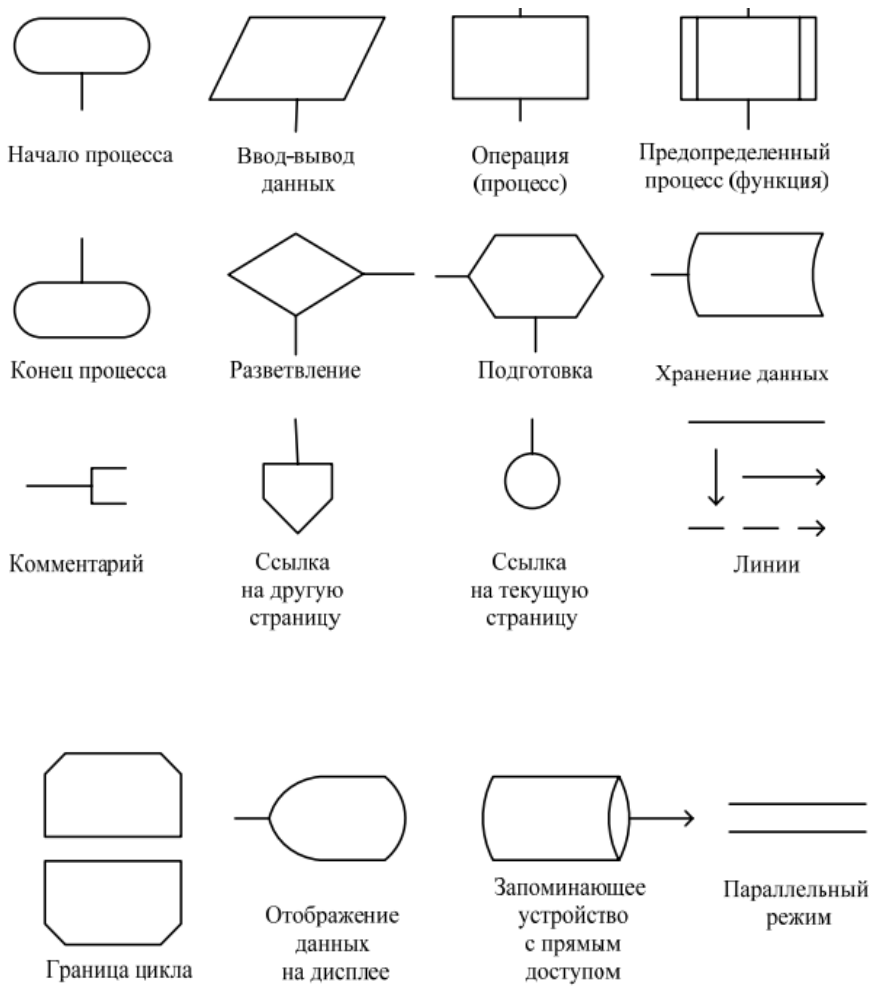

Рисунок 1 - Условные обозначения, используемые в блок-схемах

Для стандартизации и унификации языка схем алгоритмов в 1992 г. был принят ГОСТ 19.701–90 «Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения». В настоящее время данный стандарт продолжает действовать в Республике Беларусь.

На рисунке 2 приведена блок-схема алгоритма нахождения максимального из трех заданных чисел, которая использует разветвляющийся алгоритм.

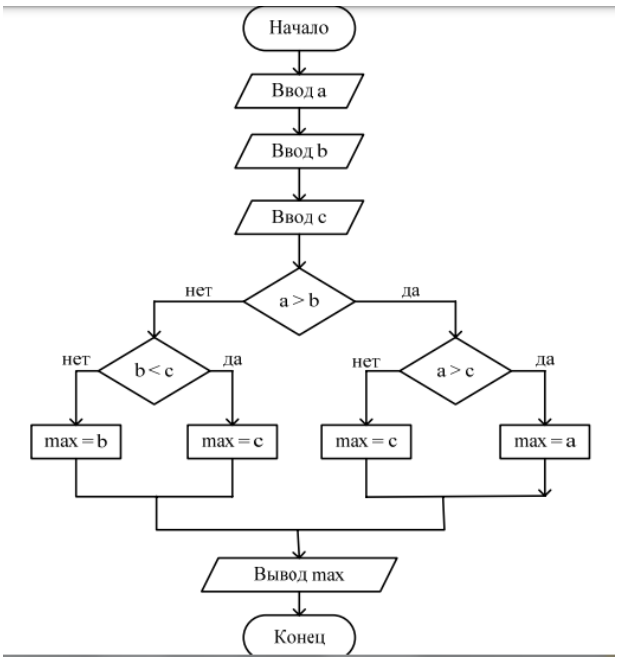

Рисунок 2 - Блок-схема алгоритма нахождения максимального числа

На рисунке 3 приведена блок-схема алгоритма вычисления графика функции в заданном интервале, в которой используется циклический вычислительный процесс с известным числом повторений.

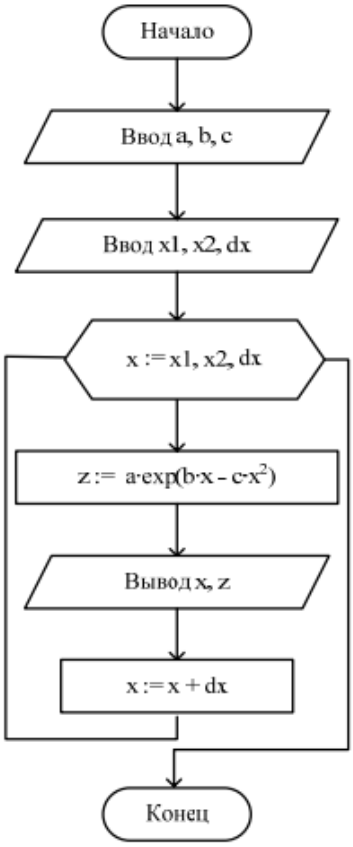

Рисунок 3 - Блок-схема алгоритма вычисления графика функции в заданном интервале

Рассмотрим еще один **пример**, в котором используется цикл с неизвестным числом повторений. Необходимо вычислить значение функции Y = sinx через разложение функции в бесконечный ряд: Y = sinx = x – x 3 / 3! + x 5 / 5! – x 7 / 7! + … . Вычисления прекращаются, когда разность между модулями двух соседних слагаемых станет меньше величины e = 0,0001. Схема алгоритма решения данной задачи имеет вид, представленный на рисунке 4. Как видно, здесь проверка условий окончания циклических вычислений осуществляется в конце цикла.

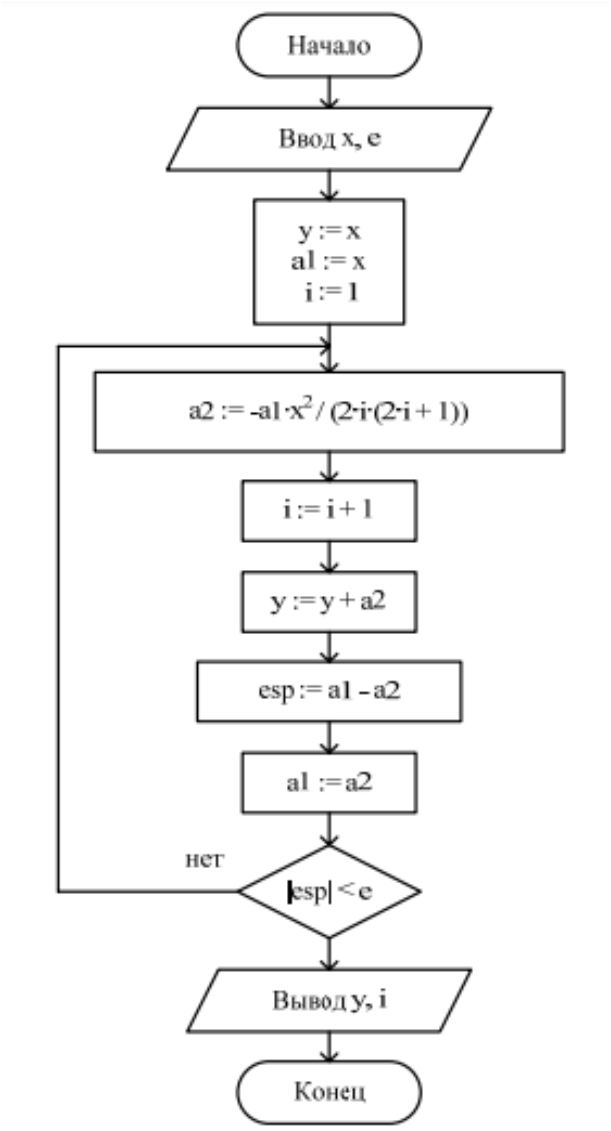

Рисунок 4 - Блок-схема алгоритма вычисления значения функции с переменным количеством шагов

Понятия алгоритма и программы не имеют четкого разграничения. Так, программа, записанная на алгоритмическом языке – это окончательный вариант алгоритма решения задачи, ориентированный на конкретного исполнителя (компьютер или язык программирования).

Тема 1.1. Структура программного модуля. Состав интегрированной программной среды. Типы данных

Алфавит

В тексте на любом естественном языке можно выделить 4 элемента:

1) символы;
2) слова;
3) словосочетания;
4) предложения.

В языках программирования:

1) лексемы (слова);
2) выражения (словосочетания);
3) операторы (предложения) или инструкции, команды.

Из символов образуются лексемы, из лексем – выражения, из выражений и лексем – операторы.

Символы, образующие алфавит языка, – это, основные неделимые знаки, используемые в данном языке.

Используемые символы

Множество символов используемых в языке СИ можно разделить на пять групп:

1. Символы, используемые для образования ключевых слов и идентификаторов. В эту группу входят прописные и строчные буквы английского алфавита, а также символ подчеркивания. Следует отметить, что одинаковые прописные и строчные буквы считаются различными символами, так как имеют различные коды.

2. Группа прописных и строчных букв русского алфавита и арабские цифры.

3. Знаки нумерации и специальные символы. Эти символы используются с одной стороны для организации процесса вычислений, а с другой - для передачи компилятору определенного набора инструкций.

Символ	Наименование	Символ	Наименование
,	запятая	)	круглая скобка правая
.	точка	(	круглая скобка левая
;	точка с запятой	}	фигурная скобка правая
:	двоеточие	{	фигурная скобка левая
?	вопросительный знак	<	меньше
'	апостроф	>	больше
!	восклицательный знак	[	квадратная скобка
|	вертикальная черта	]	квадратная скобка
/	дробная черта	#	номер
\	обратная черта	%	процент
~	тильда	&	амперсанд
*	звездочка	^	логическое не
+	плюс	=	равно
-	мину	"	кавычки

4. Управляющие и разделительные символы. К той группе символов относятся: пробел, символы табуляции, перевода строки, возврата каретки, новая страница и новая строка. Эти символы отделяют друг от друга объекты, определяемые пользователем, к которым относятся константы и идентификаторы. Последовательность разделительных символов рассматривается компилятором как один символ (последовательность пробелов).

5. Кроме выделенных групп символов в языке СИ широко используются так называемые, управляющие последовательности, т.е. специальные символьные комбинации, используемые в функциях ввода и вывода информации. Управляющая последовательность строится на основе использования обратной дробной черты (\) (обязательный первый символ) и комбинацией латинских букв и цифр.

Управляющая последовательность	**Наименование**	**Шеснадцатеричная замена**
\a	Звонок	007
\b	Возврат на шаг	008
\t	Горизонтальная табуляция	009
\n	Переход на новую строку	00A
\v	Вертикальная табуляция	00B
\r	Возврат каретки	00C
\f	Перевод формата	00D
\"	Кавычки	022
\'	Апостроф	027
\0	Ноль-символ	000
\\	Обратная дробная черта	05C
\ddd	Символ набора кодов в восьмеричном представлении	
\xddd	Символ набора кодов в шестнадцатеричном представлении	

Последовательности вида \ddd и \xddd (здесь d обозначает цифру) позволяет представить символ из набора кодов ПЭВМ как последовательность восьмеричных или шестнадцатеричных цифр соответственно. Например символ возврата каретки может быть представлен различными способами:

\r - общая управляющая последовательность,

\015 - восьмеричная управляющая последовательность,

\x00D - шестнадцатеричная управляющая последовательность.

Следует отметить, что в строковых константах всегда обязательно задавать все три цифры в управляющей последовательности. Например, отдельную управляющую последовательность \n (переход на новую строку) можно представить как \010 или \xA, но в строковых константах необходимо задавать все три цифры, в противном случае символ или символы следующие за управляющей последовательностью будут рассматриваться как ее недостающая часть. Например:

"ABCDE\x009FGH" данная строковая команда будет напечатана с использованием определенных функций языка СИ, как два слова ABCDE FGH, разделенные 8-ю пробелами, в этом случае если указать неполную управляющую строку"ABCDE\x09FGH",то на печати появится ABCDE=|=GH, так как компилятор воспримет последовательность \x09F как символ "=+=".

Отметим тот факт, что, если обратная дробная черта предшествует символу не являющемуся управляющей последовательностью (т.е. не включенному в табл.4) и не являющемуся цифрой, то эта черта игнорируется, а сам символ представляется как литеральный. Например:

символ \h представляется символом h в строковой или символьной константе.

Кроме определения управляющей последовательности, символ обратной дробной черты (\) используется также как символ продолжения. Если за (\) следует (\n), то оба символа

игнорируются, а следующая строка является продолжением предыдущей. Это свойство может быть использовано для записи длинных строк.

Константы

Константами называются перечисление величин в программе. В языке СИ разделяют четыре типа констант: целые константы, константы с плавающей запятой, символьные константы и строковыми литералы.

Целая константа: это десятичное, восьмеричное или шестнадцатеричное число, которое представляет целую величину в одной из следующих форм: десятичной, восьмеричной или шестнадцатеричной.

Десятичная константа состоит из одной или нескольких десятичных цифр, причем первая цифра не должна быть нулем (в противном случае число будет воспринято как восьмеричное).

Восьмеричная константа состоит из обязательного нуля и одной или нескольких восьмеричных цифр (среди цифр должны отсутствовать восьмерка и девятка, так как эти цифры не входят в восьмеричную систему счисления).

Шестнадцатеричная константа начинается с обязательной последовательности 0x или 0X и содержит одну или несколько шестнадцатеричных цифр (цифры представляющие собой набор цифр шеснадцатеричной системы счисления: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)

Примеры целых констант:

Десятичная константа	Восьмеричная константа	Шестнадцатеричная константа
16	020	0x10
127	0117	0x2B
240	0360	0XF0

Если требуется сформировать отрицательную целую константу, то используют знак "-" перед записью константы (который будет называться унарным минусом).

Например: -0x2A, -088, -16 .

Каждой целой константе присваивается тип, определяющий преобразования, которые должны быть выполнены, если константа используется в выражениях. Тип константы определяется следующим образом:

- десятичные константы рассматриваются как величины со знаком, и им присваивается тип int (целая) или long (длинная целая) в соответствии со значением константы. Если константа меньше 32768, то ей присваивается тип int в противном случае long.
- восьмеричным и шестнадцатеричным константам присваивается тип int, unsigned int (беззнаковая целая), long или unsigned long в зависимости от значения константы согласно:

Диапазон шестнадцатеричных констант	Диапазон восьмеричных констант	Тип
0x0 - 0x7FFF	0 - 077777	int
0X8000 - 0XFFFF	0100000 - 0177777	unsigned int
0X10000 - 0X7FFFFFFF	0200000 - 017777777777	long
0X80000000 - 0XFFFFFFFF	020000000000 - 037777777777	unsigned long

Для того чтобы любую целую константу определить типом long, достаточно в конце константы поставить букву "l" или "L".

Пример:
5l, 6l, 128L, 0105L, OX2A11L.

Константа с плавающей точкой - десятичное число, представленное в виде действительной величины с десятичной точкой или экспонентой.

Формат имеет вид:
[цифры].[цифры] [E|e [+|-] цифры] .

Число с плавающей точкой состоит из целой и дробные части и (или) экспоненты. Константы с плавающей точкой представляют положительные величины удвоенной точности (имеют тип double). Для определения отрицательной величины необходимо сформировать константное выражение, состоящее из знака минуса и положительной константы.

Примеры: 115.75, 1.5E-2, -0.025, .075, -0.85E2

Символьная константа - представляется символом заключенном в апострофы. Управляющая последовательность рассматривается как одиночный символ, допустимо ее использовать в символьных константах. Значением символьной константы является числовой код символа.

Примеры:
' '- пробел,
'Q'- буква Q,
'\n' - символ новой строки,
'\\' - обратная дробная черта,
'\v' - вертикальная табуляция.

Символьные константы имеют тип int и при преобразовании типов дополняются знаком.

Строковая константа (литерал) - последовательность символов (включая строковые и прописные буквы русского и латинского а также цифры) заключенные в кавычки (").

Например: "Школа N 35", "город Тамбов", "YZPT КОД".

Отметим, что все управляющие символы, кавычка ("), обратная дробная черта (\) и символ новой строки в строковом литерале и в символьной константе представляются соответствующими управляющими последовательностями. Каждая управляющая последовательность представляется как один символ.

Например, при печати литерала "Школа \n N 35" его часть "Школа" будет напечатана на одной строке, а вторая часть "N 35" на следующей строке.

Символы строкового литерала сохраняются в области оперативной памяти. В конец каждого строкового литерала компилятором добавляется нулевой символ, представляемый управляющей последовательностью \0.

Строковый литерал имеет тип char[]. Это означает, что строка рассматривается как массив символов. Отметим важную особенность, число элементов массива равно числу символов в строке плюс 1, так как нулевой символ (символ конца строки) также является элементом массива. Все строковые литералы рассматриваются компилятором как различные объекты. Строковые литералы могут располагаться на нескольких строках. Такие литералы формируются на основе использования обратной дробной черты и клавиши ввод. Обратная черта с символом новой строки игнорируется компилятором, что приводит к тому, что следующая строка является продолжением предыдущей.

Например:
"строка неопределенной \n
длины"
полностью идентична литералу
"строка неопределенной длинны".

Для сцепления строковых литералов можно использовать символ (или символы) пробела. Если в программе встречаются два или более строковых литерала, разделенные только пробелами,

то они будут рассматриваться как одна символьная строка. Этот принцип можно использовать для формирования строковых литералов занимающих более одной строки.

Идентификатор

Идентификатором называется последовательность цифр и букв, а также специальных символов, при условии, что первой стоит буква или специальный символ. Для образования идентификаторов могут быть использованы строчные или прописные буквы латинского алфавита. В качестве специального символа может использоваться символ подчеркивание (_). Два идентификатора, для образования которых используются совпадающие строчные и прописные буквы, считаются различными.

Например: abc, ABC, A128B, a128b .

Важной особенностью является то, что компилятор допускает любое количество символов в идентификаторе, хотя значимыми являются первые 31 символ. Идентификатор создается на этапе объявления переменной, функции, структуры и т.п. после этого его можно использовать в последующих операторах разрабатываемой программы. Следует отметить важные особенности при выборе идентификатора.

Во первых, идентификатор не должен совпадать с ключевыми словами, с зарезервированными словами и именами функций библиотеки компилятора языка СИ.

Во вторых, следует обратить особое внимание на использование символа (_) подчеркивание в качестве первого символа идентификатора, поскольку идентификаторы построенные таким образом, что, с одной стороны, могут совпадать с именами системных функций и (или) переменных, а с другой стороны, при использовании таких идентификаторов программы могут оказаться непереносимыми, т.е. их нельзя использовать на компьютерах других типов.

В третьих, на идентификаторы используемые для определения внешних переменных, должны быть наложены ограничения, формируемые используемым редактором связей (отметим, что использование различных версий редактора связей, или различных редакторов накладывает различные требования на имена внешних переменных).

Хранение данных в оперативной памяти

Память компьютера можно рассматривать как ряд ячеек. Все ячейки последовательно пронумерованы. Эти номера называют адресами памяти. Переменная занимает одну или несколько ячеек, в которых можно хранить некоторое значение.

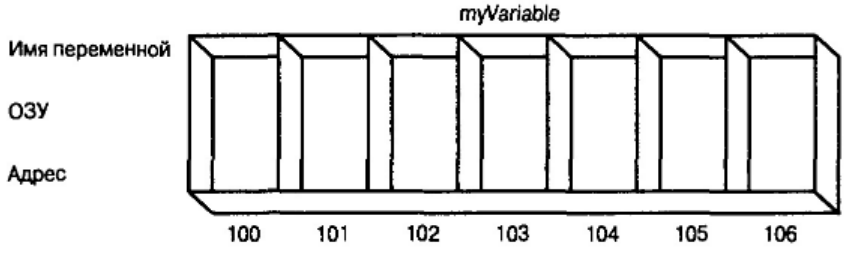

Рисунок 1.1 Схематическое представление памяти

Рисунок 1.1 Схематическое представление памяти

Имя переменной (например, MyVariable) можно представить себе как надпись на ячейке памяти, по которой, не зная настоящего адреса памяти, можно ее найти. На рис. 1.3 схематически представлена эта идея. Согласно этому рисунку, переменная MyVariable начинается с ячейки с адресом 103. В зависимости от своего размера, переменная MyVariable может занимать одну или несколько ячеек памяти.

Резервирование памяти

При определении переменной в языке С++ необходимо предоставить компилятору информацию о ее типе, например int, char или другого типа. Благодаря этой информации компилятору будет известно, сколько места нужно зарезервировать для нее и какого рода значение будут хранить в этой переменной.

Каждая ячейка имеет размер в один байт. Если для переменной указанного типа требуется четыре байта, то для нее будет выделено четыре ячейки, т.е. именно по типу переменной (например, int) компилятор судит о том, какой объем памяти (сколько ячеек) нужно зарезервировать для этой переменной.

Для переменных одних и тех же типов на компьютерах разных марок может выделяться разный объем памяти, в то же время в пределах одного компьютера две переменные одинакового типа всегда будут иметь постоянный размер.

Не прекращаются споры о произношении имени типа char. Одни произносят его как "кар", другие — как "чар". Поскольку это сокращение слова character, то первый вариант правильнее, но вы вольны произносить его, так как вам удобно.

Ключевые слова

Ключевые слова - это зарезервированные идентификаторы, которые наделены определенным смыслом. Их можно использовать только в соответствии со значением известным компилятору языка СИ.

Приведем список ключевых слов

auto double int struct break else long switch
register tupedef char extern return void case float
unsigned default for signed union do if sizeof
volatile continue enum short while

Кроме того в рассматриваемой версии реализации языка СИ, зарезервированными словами являются:

_asm, fortran, near, far, cdecl, huge, paskal, interrupt .

Ключевые слова far, huge, near позволяют определить размеры указателей на области памяти. Ключевые слова _asm, cdelc, fortran, pascal служат для организации связи с функциями написанными на других языках, а также для использования команд языка ассемблера непосредственно в теле разрабатываемой программы на языке СИ.

Ключевые слова не могут быть использованы в качестве идентификаторов.

Использование комментариев в тексте программы

Комментарий - это набор символов, которые игнорируются компилятором, на этот набор символов, однако, накладываются следующие ограничения. Внутри набора символов, который представляет комментарий не может быть специальных символов определяющих начало и конец комментариев, соответственно (/* и */). Отметим, что комментарии могут заменить как одну строку, так и несколько. Комментарии в языке С++ обозначаются «//».

Например:

```
/* комментарии к программе */
/* начало алгоритма */
 или
/* комментарии  можно  записать в следующем виде,  однако надо
 быть  осторожным,  чтобы  внутри  последовательности,  которая  игнорируется
компилятором не попались операторы программы, которые также будут игнорироваться */
```

Неправильное определение комментариев.

/* комментарии к алгоритму /* решение краевой задачи */ */
или
/* комментарии к алгоритму решения */ краевой задачи */

Типы данных

В языке C++ имеется две группы типов данных: основные и производные. Основные типы данных служат для представления целых чисел и чисел с плавающей точкой. К производным типам данных относятся массивы, строки, указатели и структуры.

Основные типы целочисленных данных в C++: char, int, short, long. Каждый из этих типов данных подразделяется на две разновидности: со знаком и без знака (signed, unsigned).

Данными типа char являются различные символы, причем значением этих символов является численное значение во внутренней кодировке ЭВМ. Данные этого типа занимают 1 байт и меняются в диапазоне:

signed char от -128..127

unsigned char от 0..255.

Отметим, что если необходимо иметь дело с переменными, принимающими значения русских букв, то их тип должен быть unsigned char, т.к. коды русских букв больше 127.

Базовые типы переменных, используемые в программах C++, представлены в табл. 1.2. В ней также приведены обычные размеры переменных указанных типов и предельные значения, которые могут храниться в этих переменных.

Таблица 1.2 Типы переменных

Тип	Размер	Значение
bool	1 байт	rue или false
unsigned short int	2 байта	от 0 до 65 535
short int	2 байта	от -32 768 до 32 767
unsigned long int	4 байта	от 0 до 4 294 967 295
long int	4 байта	от - 2 147 483 648 до 2 147 483 647
int (16 разрядов)	2 байта	от -32 768 до 32 767
int (32 разраяда)	4 байта	от - 2 147 483 648 до 2 147 483 647
unsigned int (16 разрядов)	2 байта	от 0 до 65 535
unsigned int (32 разряда)	4 байта	от 0 до 4 294 967 295
char	1 байт	256 значение символов
float	4 байта	от 1.2e-38 до 3.4e38
double	8 байтов	от 2.2e-308 до 1.8e308

Иногда программам приходится работать с символьными данными, величины которых больше, чем определяется восьмибитовым байтом (японский набор символов).

Справиться с такой ситуацией в языке C++ можно двумя способами:

- если большой набор символов является основным набором символов для данной реализации, то поставщик компилятора может определить, что размер данных типа char равен 16 битам или больше.

- реализация языка может допускать работу, как с основным, так и с расширенным набором символов. Обычные 8 битовые данные типа char могут использоваться для представления основного набора символов, а данные нового типа wchar_t – для представления расширенного набора символов. Данные типа wchar_t являются целочисленными данными, имеющие достаточную величину для представления самого большого расширенного набора символов, используемого в данной системе (набор символов unicode).

В самом стандарте языка C++ определено, что sizeof (char)=1

sizeof (char) <= sizeof (short) <= sizeof (int) <= sizeof (long)

sizeof (type) – операция определения размера type в байтах, т.е. величина данных типа short не меньше 16 битов, типа int не меньше размера данных типа short, величина типа данных типа long не меньше 32 битов и не меньше размера данных типа int.

Для каждого из рассмотренных типов имеется его разновидность – тип данных без знака. Данные этих типов не могут принимать отрицательные значения. Преимуществом здесь является то, что увеличена максимально возможное значение данных.

В настоящее время во многих системах используется минимально гарантированные размеры данных, и данные типа short имеют величину =16битам, данные типа long = 32 битам. Для данных типа int остается возможность выбора, их величина может быть = 16, 24 или 32 битам и при этом соответствовать стандарту.

Если отсутствует причина заставляющая выбирать какой-либо иной тип данных, то используйте int. Если переменная представляет величину, которая никогда не может быть отрицательной, то можно использовать тип данных без знака.

Если вы знаете, что переменная может принимать значение, превышающее 16 битовое целое число, то используйте тип данных long. Поступайте так даже в том случае, если данные типа int имеют величину 32 бита.

Числа с плавающей точкой хранятся в виде двух составных частей. Одна часть представляет некоторое значение, а вторая – степень этого значения. Степень служит для перемещения десятичной точки. Отсюда и термин «плавающая точка». В языке C++ имеются два способа записи чисел с плавающей точкой:

- стандартная запись чисел с десятичной точкой. Даже если дробная часть равна 0, число будет храниться в формате с плавающей точкой, а не целых чисел.

- экспоненциальная форма записи чисел: 3.45E6 – здесь 6 – экспонента, а 3.45 – мантисса («+» - умножение, «-» - деление)

$3.45E+6 = 3.45*10^6$.

В C++ имеется три типа данных с плавающей точкой: Float, double, long double. Данные этих типов характеризуются числом значащих цифр, которые они могут иметь и минимально допустимым диапазоном значений экспоненты. В результате требований, предъявляемых в языках C и C++ к числу значащих цифр, данные типа Float имеют минимальную величину 32 бита, данные типа double – минимальную величину 48 битов, и данные типа long double имеют величину не меньшую, чем данные типа double. Все эти три типа могут иметь и одинаковые размеры.

Определение переменной

Чтобы создать или определить переменную, нужно указать ее тип, за которым (после одного или нескольких пробелов) должно следовать ее имя, завершающееся точкой с запятой. Для имени переменной можно использовать практически любую комбинацию букв, но оно не должно содержать пробелов, например: x, J23qrsnf и myAge. Хорошими считаются имена, позволяющие судить о назначении переменных, ведь удачно подобранное имя способно облегчить понимание работы программы в целом. В следующем выражении определяется целочисленная переменная с именем myAge:

int myAge;

При объявлении переменной для нее выделяется (резервируется) память. Резервирование памяти не очищает ячейки от значений, которые ранее в них хранились, поэтому если за объявлением переменной не следует ее инициализация, то текущее значение этой переменной будет непредсказуемым, а не нулевым, как думают многие.

Уважающие себя программисты стремятся избегать таких нечитабельных имен переменных, как J23qrsnf, а однобуквенные имена (например, x или i) используют только для временных переменных, таких как счетчики циклов. Старайтесь использовать как можно более информативные имена, например myAge или howMany. Такие имена даже три недели спустя помогут вам вспомнить, что вы имели в виду, когда писали те или иные программные строки.

Поставьте следующий эксперимент. Опираясь лишь на первые пять строк программы, попробуйте догадаться, для чего предназначены объявленные ниже переменные.

```
Пример 1
int main()
{
      unsigned short x;
```

```
      unsigned short y;
      unsigned short z;
      z = x * y;
      return 0;
}


Пример 2
int main()
{
       unsigned short Width;
       unsigned short Length;
       unsigned short Area;
       Area = Width * Length;
       return 0;
}
```

Если вы скомпилируете эту программу, компилятор выведет предупреждение о том, что эти переменные не инициализированы.

Соглашения об именовании

Существуют несколько соглашений по именам переменных. Хоть и не так уж и важно, каких именно принципов придерживаться, желательно оставаться верным им, по крайней мере, на протяжении работы над одним проектом.

Многие программисты предпочитают использовать для имен переменных исключительно маленькие буквы. Если для имени переменной необходимы два слова (например, my car), то в соответствии с наиболее популярными соглашениями возможны два варианта: my_car и myCar. Последняя форма записи называется "верблюжьей", поскольку одна прописная буква посередине слова напоминает горб верблюда.

Большинство профессиональных программистов применяют "венгерскую нотацию", отличительной чертой которой является префикс, указывающий на тип переменной. Так, имена целочисленных переменных (типа int) должны начинаться со строчной буквы i, длинные целые (типа long int) - со строчной буквы l. Соответствующими префиксами должны быть отмечены константы, глобальные переменные, указатели и другие объекты. Такая форма записи характерна скорее для языка C, чем для C++.

Присвоение значений переменным

С этой целью используется оператор присваивания (=). Так, чтобы присвоить число 5 переменной Width, запишите следующее:

unsigned short Width;

Width = 5;

Тип long — это сокращенное название типа long int, a short— сокращенное название типа short int.

Эти две строки можно объединить в одну и инициализировать переменную Width в процессе определения:

unsigned short Width = 5;

Инициализация напоминает присваивание, особенно в случае инициализации целочисленных переменных. Ниже, при рассмотрении констант, вы узнаете, что некоторые значения обязательно должны быть инициализированы, поскольку по отношению к ним нельзя выполнять операцию присваивания. Существенное отличие инициализации от присваивания состоит в том, что она происходит в момент создания переменной.

Подобно тому, как можно определять сразу несколько переменных, можно и инициализировать сразу несколько переменных при их создании. Например:

```
// создаем две переменных типа long и инициализируем их
long width = 5, length = 7;
```

В этом примере переменная width типа long int была инициализирована значением 5, а переменная length того же типа — значением 7. При определении нескольких переменных в одной строке, инициализировать можно только некоторые из них:

```
int myAge = 39, yourAge, hisAge = 40;
```

В этом примере создаются три переменных типа int, а инициализируются только первая и третья.

Ключевое слово typedef

Порой бывает утомительно и скучно многократно писать такие слова, как unsigned short int. В языке C++ для этой фразы предусмотрена возможность создания псевдонима с помощью ключевого слова typedef, которое означает определение типа.

Фактически создается синоним; не следует путать это с созданием нового типа. Ключевое слово typedef применяется следующим образом:

```
typedef unsigned short int USHORT;
```

- создает новый псевдоним USHORT, который можно использовать везде, где нужно было бы написать фразу unsigned short int

// Простая программа на C++

```
# include <iostream>
Using namespace std;

Int main()
{cout << “Добро пожаловать в C++! \n”;
Return 0;
}
```

Программа C++ строится из отдельных блоков, называемых функциями. Как правило, программа разделяется на ряд крупных задач, а затем для выполнения этих задач разрабатываются отдельные функции. Данная программа достаточно проста и содержит следующие элементы: комментарии, на которые указывает «//», директиву препроцессора include, директиву Using namespace, заголовок функции Int main(), тело функции ограниченное символами {}, оператор, в котором для вывода сообщения на экран используется объект cout, оператор Return, завершающий выполнение функции main().

Функция main()

В своей структуре эта программа имеет следующую общую структуру:

```
Int main()
{
Операторы;
Return 0;
}
```

Все эти строки вместе составляют определение функции. Это определение состоит из двух частей:

- заголовка функции (Int main());

- тела функции, заключенного в {}.

Заголовок функции в краткой форме определяет интерфейс между функцией и остальной частью программы, а тело функции определяет то, что собственно делает функция.

В языке C++ каждая законченная инструкция называется оператором. Каждый оператор должен заканчиваться «;». Заключительный оператор функции main() называется оператором возврата.

В общем случае функция C++ активизируется или вызывается другой функцией, а заголовок некоторой функции описывает интерфейс между этой функцией и той, которая ее вызывает. Слово, стоящее перед именем функции, называется возвращаемым типом функции. Оно описывает информацию, передаваемую из функции назад в ту функцию, которая ее вызвала. Информация в круглых скобках, следующих за именем функции, называется списком аргументов или списком параметров. Этот список описывает информацию, передаваемую из вызывающей в вызываемую функцию.

Как правило, функция main() вызывается кодом начальной загрузки, который добавляется в программу компилятора для связи программы с ОС. По существу заголовок функции описывает интерфейс между main() и ОС.

Заголовок Int main() означает, что функция main() может возвращать целочисленное значение, и не принимать от нее никакой информации.

Препроцессор C++ и файл iostream

```
 # include <iostream>
Using namespace std;

# include <iostream.h>
```

При создании исполняемого кода программ C++ используется препроцессор – это программа, которая обрабатывает исходный файл перед основной компиляцией.

include <iostream> - эта директива приводит к тому, что препроцессор добавляет в программу содержание файла iostream. Это типичное для препроцессора действие: добавление или изменение текста в исходном коде перед компиляцией. Необходимость добавления содержимого файла iostream связана с передачей данных между программой и внешним миром.

Схема ввода/вывода в языке C++ включает в себя несколько определений, которые находятся в файле iostream. В нашей первой программе необходимы эти определения, чтобы отобразить на экране сообщение.

Директива # include приводит к тому, что содержимое файла iostream передается в компилятор вместе с содержимым исходного файла. В сущности содержимое файла iostream заменяет в программе строку: # include <iostream>.

Такие файлы, как iostream, называются файлами включения (поскольку они включаются в другие) или заголовочными файлами (поскольку они включаются в начало файла).

Заголовочные файлы в языке C имеют расширение h – это самый простой способ идентификации типа файла по его имени.

Например, файл **math.h** поддерживает различные математические функции языка C.

В языке C++ первоначально была принята такая же система наименования. Однако, позднее способ именования заголовочных файлов в C++ изменили. Теперь расширение h зарезервировано для старых заголовочных файлов языка C, а заголовочные файлы C++ вообще не имеют расширения. Ряд заголовочных файлов языка C был преобразован в заголовочные файлы C++. Эти файлы были переименованы: у них убрали расширение (чтобы имена были в стиле языка C++) и добавили префикс «c», указывающий, что они поддерживают элементы языка C.

Пример, math.h – C;

cmath – C++.

Области имен

Если вместо файла iostream.h вы используете файл iostream, то для того, чтобы определения в этом файле были доступны вашей программе необходимо использовать директиву для области имен:

Using namespace std;

Она называется директивой Using.

Область имен – это новая особенность языка C++ предназначенная для упрощения создания программ, в которой объединяются готовые коды программ от разных поставщиков. Одна из возможных проблем заключается в том, что в двух используемых пакетах программных продуктов могут находиться функции с одинаковыми именами. Такое средство, как область имен дает возможность поставщику помещать свои продукты в программную единицу (блок, элемент), называемую областью имен, а программисту использовать имена из этой области, чтобы указывать: какой именно программный продукт ему требуется. Точно также классы, функции и переменные, являющиеся стандартными компонентами компилятора в C++ помещаются в область имен, которая называется std. Так делается в файлах без расширения. Это означает, что переменная cout, которая используется для вывода данных и определение которое находится в файле iostream, в действительности называется: std:: cout.

ТЕМА 1.1 ОПЕРАЦИИ ВЫБОРА. ЛИНЕЙНЫЕ АЛГОРИТМЫ И ПРОГРАММЫ. ФОРМИРОВАНИЕ ПРОЕКТА

Операции

В языке C++ операторы управляют последовательностью выполнения выражений, возвращают результаты вычислений или ничего не делают (пустые операторы). Все выражения в C++ оканчиваются точкой с запятой. Пустые операторы представляет собой просто точку с запятой. Наиболее простой пример выражения — это операция присвоения значения:

x = a + b;

В отличие от алгебры, это выражение не означает, что x равняется a+b. Данное выражение следует понимать так: присвоим результат суммирования значений переменных a и b переменной x, или присвоим переменной x значение a+b. Несмотря на то, что в этом выражении выполняется сразу два действия — вычисление суммы и присвоение значения, после выражения устанавливается только один символ точки с запятой.

Оператор (=) присваивает результаты операций, выполняемых над операндами, расположенными справа от знака равенства, операнду, находящемуся слева от него.

Символы пробелов, к которым относятся не только пробелы, но и символы табуляции и разрыва строки, в выражениях обычно игнорируются.

Блоки кода и составные операторы

Иногда для облегчения восприятия программы логически взаимосвязанные выражения удобно объединять в комплексы, называемые блоками. Блок начинается открывающей фигурной скобкой ({) и оканчивается закрывающей фигурной скобкой (}). Хотя каждое выражение в блоке должно оканчиваться точкой с запятой, после символов открытия и закрытия блока точки с запятой не ставятся, как в следующем примере:

```
{
   temp = a;
   a = b;
   b = temp;
}
```

Этот блок выполняется как одно выражение, осуществляющее обмен значениями между a и b.

Выражения

Все, что обрабатывает значения в C++, называет выражением. Все выражения возвращают значение. Так, операция 3+2; возвращает значение 5, т.е. является выражением. Все выражения являются операторами.

Большинство участков кода, как это ни удивительно, тоже являются выражениями. Приведем лишь три примера:

```
3.2 // возвращает значение 3.2
PI // вещественная константа, которая возвращает значение 3.14
SecondsPerMinute // целочисленная константа, которая возвращает 60
```

Предполагая, что PI — константа, равная 3.14. а SecondsPerMinute — константа, равная 60, можно утверждать, что все эти операторы являются выражениями.

Операции всегда располагаются справа от оператора присваивания:

```
y = x = a + b;
```

Данное выражение выполняет представленную ниже последовательность действий:

– Прибавляем a к b.
– Присваиваем результат выражения a + b переменной x.
– Присваиваем результат выражения присваивания x = a + b переменной y.

Если переменные a, b, x и y являются целыми и если a имеет значение 2, а b — значение 5, то переменным x и y будет присвоено значение 7.

Операторы

Оператор — это символ, который заставляет компилятор выполнять некоторое действие. Операторы воздействуют на операнды. Операндами в C++ могут быть как отдельные литералы, так и целые выражения. Язык C++ располагает двумя видами операторов:

1) операторы присваивания;
2) математические операторы;
3) операторы отношений;
4) логические операторы;

Оператор присваивания

Оператор присваивания (=) позволяет заменить значение операнда, расположенного с левой стороны от знака равенства, значением, вычисляемым с правой стороны от него. Так, выражение

```
x = a + b;
```

присваивает операнду x значение, которое является результатом сложения значений переменных a и b.

Операнд, который может находиться слева от оператора присваивания, называется адресным операндом, или l-значением (от англ. слова left, т.е. левый). Операнд, который может находиться справа от оператора присваивания, называется операционным операндом, или r-значением (от англ. слова right, т.е. правый).

Константы могут быть только r-значениями и никогда не бывают адресными операндами, поскольку в ходе выполнения программы значения констант изменять нельзя. Так, можно записать:

```
x = 35; // правильно
```

Но нельзя записать:

```
35 = x; // ошибка!
```

Математические операторы

В C++ используется пять математических операторов: сложения (+), вычитания (-), умножения (*), целочисленного деления (/) и деления по модулю (%).

В операциях сложения и вычитания разобраться несложно: они возвращают сумму и разность двух операндов. Хотя следует отметить, что вычитание беззнаковых целых может привести к удивительным результатам, если полученная разность окажется отрицательным числом.

Целочисленное деление несколько отличается от обычного. При делении числа 21 на число 4 (21/4) в случае целочисленного деления в ответе получается 5 и остаток 1.

Чтобы получить остаток, нужно число 21 разделить по модулю 4 (21 % 4), в результате получим остаток 1.

Операция деления по модулю иногда оказывается весьма полезной, например, если вы захотите вывести из ряда чисел каждое десятое значение. Любое число, результат деления которого по модулю 10 равен нулю, является кратным десяти, т.е. делится на 10 без остатка. Так, результат выражения 1 % 10 равен 1; 2 % 10 равен 2 и т.д.; а 10 % 10 равен 0. Результат от деления 11 % 10 снова равен1; 12 % 10 снова равен 2; и так можно продолжать до следующего числа, кратного 10, которым окажется 20.

Совместное использование математических операторов с операторами присваивания

Нет ничего необычного в том, чтобы к переменной прибавить некоторое значение, а затем присвоить результат той же переменной. Если у вас есть переменная myAge и вы хотите увеличить ее значение на два, можно записать следующее:

```
int myAge = 5;
int temp;
```

```
temp = myAge + 2; // складываем 5 + 2 и результат помещаем в temp
myAge = temp; // значение возраста снова помещаем в myAge
```

Однако этот метод грешит излишествами. В языке C++ можно поместить одну и ту же переменную по обе стороны оператора присваивания, и тогда предыдущий блок сведется лишь к одному выражению:

```
myAge = myAge + 2;
```

В алгебре это выражение рассматривалось бы как бессмысленное, но в языке C++ оно читается следующим образом: добавить два к значению переменной myAge и присвоить результат переменной myAge.

Существует еще более простой вариант предыдущей записи, хотя его труднее читать:

```
myAge += 2;
```

Этот оператор присваивания с суммой **(+=)** добавляет r-значение к l-значению, а затем снова записывает результат в l-значение. Если бы до начала выполнения выражения переменная myAge имела значение 4, то после ее выполнения значение переменной myAge стало бы равным 6.

Помимо оператора присваивания с суммой существуют также **оператор присваивания с вычитанием (-=), делением (/=), умножением (*-) и делением по модулю (%=)**.

Инкремент и декремент

Очень часто в программах к переменным добавляется (или вычитается) единица. В языке C++ увеличение значения на 1 называется *инкрементом*, а уменьшение на 1 — *декрементом*. Для этих действий предусмотрены специальные операторы.

Оператор инкремента (++) увеличивает значение переменной на 1, а оператор декремента (--) уменьшает его на 1. Так, если у вас есть переменная C и вы хотите прирастить ее на единицу, используйте следующее выражение:

```
C++;        // Увеличение значения C на единицу
C = C + 1; // Это же выражение можно было бы записать
           //следующим образом:
C += 1;    // что, в свою очередь, равносильно выражению
```

Префикс и постфикс

Как оператор инкремента, так и оператор декремента работает в двух вариантах: *префиксном* и *постфиксном*. Префиксный вариант записывается перед именем переменной (++myAge), а постфиксный — после него (myAge++).

В простом выражении вариант использования не имеет большого значения, но в сложном при выполнении приращения одной переменной с последующим присваиванием результата другой переменной это весьма существенно. *Префиксный оператор вычисляется до присваивания, а постфиксный — после.*

Семантика префиксного оператора следующая: инкрементируем значение, а затем считываем его.

Семантика постфиксного оператора иная: считываем значение, а затем декрементируем оригинал.

На первый взгляд это может выглядеть несколько запутанным, но примеры легко проясняют механизм действия этих операторов. Если x — целочисленная переменная, значение которой равно 5, и, зная это, вы записали

```
int a = ++x;
```

то тем самым велели компилятору инкрементировать переменную x (сделав ее равной 6), а затем присвоить это значение переменной a. Следовательно, значение переменной a теперь равно 6 и значение переменной x тоже равно 6. Если же, после этого вы записали

```
int b = x++;
```

то тем самым велели компилятору присвоить переменной b текущее значение переменной x (6), а затем вернуться назад к переменной x и инкрементировать ее. В этом случае значение переменной b равно 6, но значение переменной x уже равно 7.

Приоритеты операторов

Какое действие — сложение или умножение — выполняется первым в сложном выражении, например в таком, как это:

```
x = 5 + 3 * 8;
```

Если первым выполняется сложение, то ответ равен 8*8, или 64. Если же первым выполняется умножение, то ответ равен 5 + 24, или 29.

Каждый оператор имеет значение приоритета (полный список этих значений приведен в таблице 2.1). Умножение имеет более высокий приоритет, чем сложение, поэтому значение этого "спорного" выражения равно 29.

Если два математических оператора имеют один и тот же приоритет, то они выполняются в порядке следования слева направо.

Таблица 2.1 Приоритет операторов

№	Название	Оператор
1	Область видимости	::
2	Прямое и косвенное обращение к члену класса, вызов функции, постфиксный инкремент и декремент	. -> () ++ --
3	Префиксный инкремент и декремент, инверсия и не унарные минус и плюс, получение адреса и ссылки, также функции new, new [], delete, delete [], casting, sizeof()	++ -- ^ ! - + & * ()
4	Обращение к элементу по указателю	.* ->*
5	Умножение, деление, деление по модулю	* / %
6	Сложение, вычитание	+ -
7	Сдвиг влево, сдвиг вправо	<< >>
8	Меньше, меньше или равно, больше, больше или равно	< <= > >=
9	Равно, не равно	== !=
10	Побитовое AND	&
11	Побитовое исключающее OR	^
12	Побитовое OR	|
13	Логическое AND	&&
14	Логическое OR	||
15	Условный оператор	?:
16	Операторы присвоения	= *= /= %= += -= <<= >>=
17	Запятая	,

Операторы отношений

Такие операторы используются для выяснения равенства или неравенства двух значений. Выражения сравнения всегда возвращают значения true (истина) или false (ложь). Операторы отношения представлены в табл. 2.2.

Всего в языке C++ используется шесть операторов отношений: равно (==), меньше (<), больше (>), меньше или равно (<=), больше или равно (>=) и не равно (!=).

Имя	Оператор	Пример	Значение
Равно	==	100 == 50; 50 == 50;	false true
Не равно	!=	100 != 50; 50 != 50;	true false
Больше	>	100 > 50; 50 > 50;	true false
Больше или равно	>=	100 >= 50; 50 >= 50;	true true
Меньше	<	100 < 50; 50 < 50;	false false
Меньше или равно	<=	100 <= 50; 50 <= 50;	false true

Многие начинающие программировать на языке C++ путают оператор присваивания (=) с оператором равенства (==). Случайное использование не того оператора может привести к такой ошибке, которую трудно обнаружить.

Логические операторы

Довольно часто у нас возникает необходимость проверять не одно условное выражение, а сразу несколько. Например, правда ли, что x больше y, а также что y больше z? Наша программа, прежде чем выполнить соответствующее действие, должна установить, что оба эти условия истинны либо одно из них ложно.

Представьте себе высокоорганизованную сигнальную систему, обладающую следующей логикой. Если сработала установленная на двери сигнализация И время суток после шести вечера, И сегодня НЕТ праздника ИЛИ сегодня выходной, нужно вызывать полицию. Для проверки всех условий нужно использовать три логических оператора C++;

a) Бинарное логическое **И**

Логический оператор И вычисляет два выражения, и если оба выражения возвращают true, то и оператор И также возвращает true. Если правда то, что вы голодны, И правда то, что у вас есть деньги, значит, справедливо и то, что вы можете пойти в супермаркет и купить себе что-нибудь на обед. Например, логическое выражение

```
if ( (x == 5) && (y == 5) )
```

возвратит значение true, если и обе переменные — x и y — равны 5. Это же выражение возвратит false, если хотя бы одна из переменных не равна 5. Обратите внимание, что выражение возвращает true только в том случае, если истинны обе его части.

Логический оператор И обозначается двумя символами &&. Одиночный символ & соответствует совсем другому оператору.

b) Бинарное логическое **ИЛИ**

Логический оператор ИЛИ также вычисляет два выражения. Если любое из них истинно, то и оператор ИЛИ возвращает true. Если у вас есть деньги ИЛИ у вас есть кредитная карточка, вы можете оплатить счет. При этом нет необходимости в соблюдении двух условий сразу: иметь и деньги, и кредитную карточку. Вам достаточно выполнения одного из них (хотя и то и другое — еще лучше). Например, выражение

```
if ( (x == 5) || (y == 5) )
```

возвратит значение true, если либо значение переменной x, либо значение переменной y, либо они оба равны 5.

Обратите внимание: логический оператор ИЛИ обозначается двумя символами | Оператор, обозначаемый одиночным символом |, — это совсем другой оператор.

c) унарное логическое **НЕТ**

Логический оператор НЕТ возвращает значение true, если тестируемое выражение является ложным (имеет значение false). И наоборот, если тестируемое выражение является истинным, то оператор НЕТ возвратит false! Например, выражение

```
if ( !(x == 5) )
```

возвратит значение true только в том случае, если x не равно числу 5. Это же выражение можно записать и по-другому:

```
if (x != 5)
```

d) К побитовым, или поразрядным операциям относятся:

- операция поразрядного И &;
- операция поразрядного ИЛИ | (включающее ИЛИ);
- операция поразрядного исключающего ИЛИ ^ (исключающее ИЛИ);
- унарная операция поразрядного отрицания (дополнение) ~.

Кроме того, рассматриваются операции сдвигов <<, >>.

Операнды поразрядных операций могут быть любого целого типа.

Операция & сравнивает каждый бит первого операнда с соответствующим битом второго операнда. Если оба соответствующих бита единицы, то соответствующий бит результата устанавливается в 1, в противном случае в 0.

Операция | сравнивает каждый бит первого операнда с соответствующим битом второго операнда; если любой из них или оба равны 1, то соответствующий бит результата устанавливается в 1, в противном случае в 0.

Операция ^ . Если один из сравниваемых битов равен 0, а другой равен 1, то соответствующий бит результата устанавливается в 1, в противном случае, т.е. когда оба бита равны 1 или оба равны 0, бит результата устанавливается в 0.

Операция ~ меняет в битовом представлении операнда 0 на 1, а 1 - на 0.

e) Сдвиги

Операции сдвига << >> осуществляют соответственно сдвиг влево и вправо своего левого операнда на число битовых позиций, заданных правым операндом. Таким образом, X << 2 сдвигает X влево на 2 позиции, заполняя освобождающиеся биты нулями, что эквивалентно умножению на 4. Сдвиг вправо величины без знака сопровождается дополнением старших битов нулями. Сдвиг вправо такой величины на n битов эквивалентен целочисленному делению левого операнда на 2 в степени n.

Так, 5 << 3 дает 40;

7 >> 2 дает 1.

Отметим, что правый операнд должен быть константным выражением, т.е. выражением, включающим в себя только константы. Если правый операнд отрицателен или он больше или равен числу битов левого операнда, то результат сдвига не определен. Типом результата операции сдвига является тип левого операнда.

Условный оператор

Условный оператор (?:) - это единственный оператор в языке C++, который работает сразу с тремя операндами.

Синтаксис использования условного оператора следующий:

(выражение 1) ? (выражение2) : (выражение3)

Эта строка читается таким образом: если выражение1 возвращает true, то выполняется выражение 2, в противном случае выполняется выражение3. Обычно возвращаемое значение присваивается некоторой переменной.

Преобразование типов

Явные преобразования

В языке C++ многие преобразования типов данных выполняются автоматически:

1. преобразование данных осуществляется, когда данные одного арифметического типа присваиваются переменной другого арифметического типа.

2. преобразование данных осуществляется, когда в выражении содержатся данные разных типов.

3. преобразование данных осуществляется при передаче аргументов в функции.

Разрешены любые преобразования стандартных типов одного к другому. При преобразовании более длинного типа к более короткому происходит *потеря разрядов*; при преобразовании более короткого целочисленного типа к более длинному *свободные разряды заполняются 0* (если короткий тип - беззнаковый), или происходит размножение знакового разряда (для типа со знаком).

Разрешены любые преобразования друг на друга указателей, а также ссылок. Явное преобразование типов делается посредством операции приведения типов, которая имеет две формы:

(имя_типа) операнд // Традиционная форма;

имя_типа (операнд) // функциональная форма.

Здесь **имя_типа** задаёт тип, а **операнд** является величиной, которая должна быть преобразована к заданному типу.

Отметим, что во второй форме **имя_типа** должно быть простым идентификатором.

Примеры:

```
double d = (double)5;
int i = int(d);
```

Функции преобразования типа

При целочисленном делении дробная часть результата отбрасывается. Если нам нужно получить результат, типа *double,* то хотя бы одно из двух чисел, участвующих в операции деления, должно иметь тип *double.* Если одно из двух чисел – константа, то к нему можно приписать десятичную точку и ноль. Если оба операнда деления являются целочисленными, мы получим в результате целое число, даже если переменная результата будет типа *double.* Перед тем как переменной будет присвоено значение целого числа, оно преобразуется к типу *double,* но будет уже поздно. В C++ можно потребовать, чтобы значение типа *int* было преобразовано в значение типа *double.* Имя типа *double* можно использовать точно так же, как если бы оно было именем предопределенной функции, преобразующей значения какого-нибудь типа в значения типа *double.* При таком использовании имя типа *double* выступает в роли функции. В таких случаях говорят не о функции, а о **приведении типа.** При приведении типов можно использовать и другие имена типов.

```
int  first=9, second=2;
double answer;
answer = first / second; // результат 4.0
answer = (double)first / second; // результат 4.5
```

Неявные преобразования стандартных базовых типов

Для стандартных базовых типов компилятор может выполнять любые преобразования одного типа к другому:

```
int i = 'A';              // i = 65;
char c = 256;        // Теряются 8 старших битов; c станет равно
'\0';
int j=-1;
long l=j;
long m=32768;  // Двоичное представление числа 32768 содержит
единственную единицу в 15 разряде.
int k=m;              // k = -32768, т.к. 15-й разряд для int -
знаковый.
unsigned u=m;        // u = 32768
double d = 0.999999;
```

```
long n=d;              // n = 0
```

При выполнении арифметических операций также происходит *неявное* преобразование типов. Правила здесь такие:

1) Типы char, short, enum преобразуются к типу int, a unsigned short - к unsigned int;

2) затем, если один из операндов имеет тип long double, то и второй преобразуется к long double;

3) иначе, если один из операндов имеет тип double, то и второй преобразуется к double;

4) иначе, если один из операндов имеет тип unsigned long, то и второй преобразуется к unsigned long;

5) иначе, если один из операндов имеет тип unsigned, то и второй преобразуется к unsigned;

6) иначе,если один из операндов имеет тип long, то и второй преобразуется к long;

7) иначе оба операнда имеют тип int.

Пример 1:

```
int g = 10, t = 5, t2 = t*t/2;
double s = g*t2;          // s станет равно 120;
double s0 = g*t*t/2.0;        // s0 станет равно 125.
```

Ввод и вывод

Программа на C++ может осуществлять ввод-вывод несколькими способами. Рассмотрим понятие под названием *поток*. *Входной поток* – это просто поток входных данных, которые подаются в компьютер для использования программой. Понятие поток предполагает, что способ обработки входных данных не зависит от того, откуда они приходят. Т. е. программа «видит» только поток входных данных, но не их источник. *Выходной поток* – это поток выходных данных, генерируемых программой.

Значения переменных, как и строки текста, можно выводить на экран с помощью объекта cout.

В cout - инструкции можно вставлять выражения.

```
cout << " Общая стоимость равна $ " << (price + tax);
```

Обозначение << часто называют *оператором вставки*.

```
cout << number_of_bars << "  конфет \n";
cout << one_weight << " рублей каждая \n";
```

Эти две инструкции можно записать как одну.

```
cout << number_of_bars << конфет \n" << one_weight << " рублей каждая \n";
```

Можно записать и так:

```
cout << number_of_bars << " конфет \n"
     << one_weight << " рублей каждая \n";
```

В языке C++ любая последовательность символов, заключенная в двойные кавычки, называется строкой – главным образом потому, что она состоит из нескольких символов, соединяемых вместе в более крупный блок (элемент). Символы << указывают на то, что этот оператор отправляет данную строку в объект cout; эти символы указывают направление потока информации.

Вы не должны разбивать строку на две части между кавычками; начать новую строку можно в любом месте, где можно вставить дополнительный пробел.

Esc-последовательности

Компилятор C++ распознает некоторые специальные символы, предназначенные для форматирования текста. Чтобы вставить эти символы в программу, используется знак \ , называемый символом escape последовательности, указывающая, что следующий за ней символ является управляющим. В таблице 2.3 приведены основные управляющие символы.

Символ	Его значение
\a	Оповещение (звонок)
\b	Забой
\f	Перевод страницы
\n	Новая строка
\r	Возврат каретки
\t	Табуляция
\v	Вертикальная табуляция
\'	Одиночная кавычка
\"	Двойные кавычки
\?	Вопросительный знак
\\	Наклонная черта
\000	Восьмеричное число
\xhhh	Шестнадцатеричное число

Если при выводе данных необходимо вставить пустую строку, можно просто вывести один символ новой строки \n:

```
cout << " \n ";
```

Символ **\n** рассматривается как один, который называется символом начала новой строки. При выводе строки с помощью конструкции cout автоматически переход на начало следующей строки не происходит, поэтому после первого оператора cout курсор остается на позиции, следующей за точкой в конце выведенной строки.

Если не использовать символы новой строки, то можно сделать так, чтобы последовательные операторы cout осуществляли вывод в одной строке. Например, операторы

```
cout << "The Good, the";
cout << "Bad\n";
```

отображают на экране следующие выходные данные:

The Good, theBad

Обратите внимание, что каждая очередная строка начинается непосредственно после окончания предыдущей строки. Если на стыке двух строк требуется пробел, то его необходимо включить в одну из строк.

Часто бывает необходимо вывести строку или число в определенном формате. Для этого используются так называемые *манипуляторы*.

Манипуляторы – это объекты особых типов, которые управляют тем, как iostream обрабатывает последующие аргументы. Некоторые манипуляторы могут также выводить или вводить специальные символы.

endl – при выводе перейти на новую строку;

ends – вывести нулевой байт (признак конца строки символов);

dec – выводить числа в десятичной системе (действует по умолчанию);

oct – выводить числа в восьмеричной системе;

hex – выводить числа в шестнадцатеричной системе счисления;

setw(int n) – установить ширину поля вывода в n символов (n – целое число);

setprecision(int n) – установить количество цифр после запятой при выводе вещественных чисел;

setbase(int n) – установить систему счисления для вывода чисел; n может принимать значения 0, 2, 8, 10, 16, причем 0 означает систему счисления по умолчанию, т.е. 10.

Чтобы в выходных данных указать переход на новую строку, в языке С имеется еще один способ – использование слова *endl*

```
cout << "What's next?" << endl;   // endl означает начало новой строки
```

Использовать манипуляторы просто – их надо вывести в выходной поток. Предположим, мы хотим вывести одно и то же число в разных системах счисления:

```
int x = 53;
cout << "Десятичный вид: " << dec
      << x << endl
      << "Восьмеричный вид: " << oct
      << "Шестнадцатеричный вид: "<< hex
      << x << endl
```

Аналогично используются манипуляторы с параметрами. Вывод чисел с разным количеством цифр после запятой:

```
        double x;
 // вывести число в поле общей шириной
 // 6 символов (3 цифры до запятой, десятичная точка и 2 цифры после запятой)
  cout << setw(6) << setprecision(2) << x << endl;
```

Ввод данных

Инструкция **cin** используется для ввода данных почти также, как cout – для их вывода.

```
cout << "Введите число конфет в каждом пакете \n";
cout << " и вес одной конфеты (в рублях). \n";
cout << " Затем нажмите клавишу Enter. \n";
cin >> number_of_bars;
cin >> one_weight;
```

Или так:

```
cout << "Введите число конфет в каждом пакете \n";
cout << " и вес одной конфеты (в рублях). \n";
cout << " Затем нажмите клавишу Enter. \n";
cin >> number_of_bars >> one_weight;
Можно и так:
cin >> number_of_bars
    >> one_weight;
```

Вводимые числа должны быть разделены одним или несколькими пробелами или символом новой строки.