Министерство образования и науки
Российской Федерации

Московский государственный университет геодезии и картографии

В.Н. Лениченко, В.Р. Заблоцкий

Программирование на C и C++

Учебное пособие по курсу «Информатика»

для студентов 1 курса факультета ФПКиФ

Москва, 2009 г.

УДК 519.683

Лениченко В.Н., Заблоцкий В.Р.

Программирование на С и С++.
Учебное пособие. М.: Изд-во МИИГАиК, 2009, 31с.

Учебное пособие по курсу «Информатика» предназначено для студентов первого курса, изучающих языки программирования С и С++ на факультете прикладной космонавтики и фотограмметрии МИИГАиК.

Учебное пособие рассмотрено и рекомендовано к изданию кафедрой вычислительной техники и автоматизированной обработки аэрокосмической информации и утверждено к изданию редакционно-издательской комиссией факультета прикладной космонавтики и фотограмметрии (протокол № от 16.05.2009).

Библиография: 7 назв., табл.17.

Рецензенты:

к.т.н., доцент Шайтура В.С., МИИГАиК, каф. ВТиАОИ
к.т.н., доцент Ковальчук А.К. МИПК МГТУ им. Н.Э. Баумана

1. КРАТНАЯ ИСТОРИЧЕСКАЯ СПРАВКА

Язык программирования С был создан Д. Ритчи (Dennis M. Ritchie) из Bell Laboratories в начале 1970-х годов. Язык С предоставляет программисту возможность практически полностью управлять компьютером, поскольку в нем управляющие структуры языков высокого уровня сочетаются со способностью манипулировать битами, байтами и указателями (адресами). Публикация в 1978 году книги Б. Кернигана и Д. Ритчи «Язык программирования Си» [1] превратила язык фактически в стандарт или в то, что называется «традиционный» С. Эта книга считается одной из самых удачных публикаций по языкам программирования. В 1989 году Американским национальным институтом стандартов (ANSI) был принят стандарт для языка С, так называемый С ANSI 1989. Естественно, что язык программирования С развивался и далее в 1999 году определен новый стандарт на язык С ANSI 1999.

На языке С, базируется другой язык программирования С++, созданный в начале 1980-х годов Бьярни Страуструпом (Bjarne Stroustrup), также работавшим в AT&T Bell Laboratories. С++ унаследовал все возможности С и дополнил их средствами манипулирования объектами. В 1998 году был принят ANSI/ISO-стандарт для С++. В настоящее время объективно-ориентированное программирование стало ключевой методологией программирования. С и С++ являются языками коммерческого программирования, многие коммерческие программные продукты написаны на них, включая и операционные системы.

2. ТИПЫ ДАННЫХ, ПЕРЕМЕННЫЕ И КОНСТАНТЫ

Данные могут быть представлены в программе в двух разных видах: в качестве констант, и в качестве переменных. Константы – это данные, записанные непосредственно в исходном тексте программы, например число 3.14159 или строка символов в двойных кавычках –“Leica”. Константы используются для хранения данных, но эти данные нельзя изменять. Переменные предоставляют возможность изменять данные, хранящиеся в них. Фактически переменную можно рассматривать как ячейку памяти, имеющую имя и характеризующуюся определенным размером. В эту ячейку памяти программа записывает данные, если данные изменяются, то старые значения стираются и на их место записываются новые значения. Тип переменной определяет ее размер в байтах и набор действий, разрешенных для этой переменной.

Языки С и С++ предлагают богатый ассортимент встроенных типов данных. При этом если требуемый тип данных отсутствует, его можно создать для удовлетворения практически любых запросов программиста. Далее рассматриваются различные языковые средства, связанные с типами данных, переменными и константами.

2.1 ИДЕНТИФИКАТОРЫ (НАЗВАНИЯ)

Названия (имена) переменных, функций и типов, определенных пользователем, - это примеры идентификаторов. В языках С и С++ идентификаторы представляют собой последовательности, состоящие из одной или нескольких букв английского алфавита (разрешаются как строчные, так и заглавные буквы), цифр и символа подчеркивания, например geographic_latitude или point1 . Однако идентификатор не может начинаться с цифры и содержать специальные символы. Примеры неправильных идентификаторов: точка1, scale1/10000, map_sheet N-37-1-A.

Идентификаторы могут иметь любую длину, но не все символы являются значащими. Различают два типа идентификаторов: внешние и внутренние. Внешние идентификаторы участвуют во внешнем процессе компоновки. Они называются внешними именами и включают имена функций и глобальных переменных, которые

используются в различных файлах. Если идентификатор не участвует во внешнем процессе компоновки, он является внутренним. Идентификаторы, относящиеся к этому типу, называются внутренними именами и включают, например, имена локальных переменных. В языке С значащими являются по крайней мере первые 6 символов внешнего и по крайней мере первый 31 символ внутреннего идентификатора. В языке С++ по крайней мере 1024 символа любого идентификатора являются значащими.

Символ подчеркивания часто используется для удобочитаемости, например, true_azimuth. Идентификаторы, записанные прописными и строчными буквами, распознаются как разные. Например, line и LINE — это две различные переменные. В языках С и С++ резервируются все идентификаторы, начинающиеся с двух символов подчеркивания или одного символа подчеркивания, за которым следует прописная буква.

Каждая переменная должна иметь свое уникальное имя, в одной программе не следует использовать переменные с одинаковыми именами, желательно, чтобы из названия переменной было понятно ее предназначение, т.е. давать переменным осмысленные имена.

2.2 ОБЪЯВЛЕНИЕ (ДЕКЛАРАЦИЯ) ПЕРЕМЕННЫХ

Все переменные должны быть объявлены до их использования. Чтобы объявить переменную, нужно указать ее тип, за которым после одного или нескольких пробелов записать ее имя. Общая форма объявления имеет следующий вид:

тип имя_переменной;

В конце объявления переменной должна ставиться точка с запятой. Например, чтобы объявить angle целочисленной переменной, azimuth переменной типа float и map_sheet - символьной, необходимо записать следующее.

```
int angle;
float azimuth;
char map_sheet;
```

В одной строке можно также объявить сразу несколько переменных одинакового типа, разделив их запятыми. Например, следующий оператор объявляет три переменные вещественного типа двойной точности.

```
double true_azimuth, magnetic_azimuth, grid_azimuth;
```

В программировании на С и С++ следует различать два термина: объявление и определение. В объявлении указываются имя и тип объекта. Определение предназначено для выделения памяти для этого объекта. Во многих случаях объявление также является определением. Например, объявление переменной является одновременно ее определением.

2.3 ИНИЦИАЛИЗАЦИЯ ПЕРЕМЕННЫХ

При объявлении переменной для нее выделяется память, однако ячейки памяти при этом не очищаются и содержат значения, которые ранее в них хранились. Поэтому часто используют инициализацию переменных, чтобы присвоить ей новое значение. Переменную можно инициализировать, записав после ее имени знак равенства и начальное значение. Например, следующие объявления позволяют присвоить переменной distance начальное значение 10, а переменной Quadrant начальное значение ‘+’.

```
int distance = 10;
char Quadrant = ‘+’;
```

В языке С переменные должны быть инициализированы только с использованием константных выражений. В языке С++ инициализатором может быть любое выражение, которое действительно при объявлении переменной.

2.4 КОММЕНТАРИИ

Комментарии используются программистом для создания текста, поясняющего код исходной программы. Комментарии при обработке кода программы компилятором игнорируются, поскольку они предназначены для пользователя программы, а не для компьютера. В языке С++ определены два стиля комментариев. Первый - многострочный. Он начинается с символов /* и завершается символами */. Многострочный комментарий может занимать несколько строк. Второй тип комментария - однострочный. Он начинается с символов // и завершается в конце той же строки.

В языке С поддерживается только один тип комментария - многострочный. Однако большинство компиляторов С допускают и однострочные комментарии, несмотря на их нестандартность. Далее приведен пример многострочного комментария.

*/*Вычисление горизонтального угла как среднее значение*
из двух полуприемов при КЛ и КП/*

2.5 ОСНОВНЫЕ ТИПЫ И ИХ РАЗМЕРЫ

В языке С определены следующие основные типы данных: char, int, float, double и void.

Тип	Ключевое слово
Символьный	char
Целочисленный	int
С плавающей запятой	float
С плавающей запятой двойной точности	double
Пустой	void

В переменных типа char обычно хранятся символьные данные (текстовый знак или символ), в переменных типа int - целочисленные значения, в переменных типа float и double вещественные числа. Тип void обычно используют для указания того, что функция не возвращает никакого значения.

В языке С++ к основным типам данных С добавлены еще два: bool и wcar_t. Основные типы языка С++ показаны в таблице.

Тип	Ключевое слово
Логический, или булев (true/false)	bool
Символьный	char
Целочисленный	int
С плавающей запятой	float
С плавающей запятой двойной точности	double
Пустой	void
Символьный двухбайтовый	wchar_t

Все версии С и С++ предоставляют возможность работать со следующими пятью основными типами: char, int, float, double и void. Некоторые основные типы

могут быть модифицированы с помощью одного или нескольких модификаторов типов, таких как: signed, unsigned, short, long. Модификаторы указываются перед наименованием типа, который они модифицируют. Например, для основных типов данных char и int образуются производные типы данных с помощью модификаторов типа: signed (знаковые) и unsigned (беззнаковые). Целые со знаком, например signed char или signed int, могут хранить как положительные, так и отрицательные значения. Целые без знака, например unsigned char или unsigned int могут хранить только положительные значения и нуль. Другой модификатор short (короткое целое) применим только к типу int, модификатор long применим и к типу int (например, long int) и к типу double (например, long double). Тип long double предназначен для арифметики с плавающей точкой повышенной точности.

Если при объявлении переменных используется один модификатор (без наименования типа), то предполагается использование типа int. Например, целочисленную переменную без знака можно объявить, используя лишь ключевое слово unsigned. Таким образом, следующие объявления эквивалентны.

```
unsigned int  time;   //  тип int указан явно
unsigned time;        //  здесь тип int подразумевается
```

В следующей таблице приведены все разрешенные в языках C и C++ встроенные типы данных, включая модификаторы, а также их гарантированные минимальные диапазоны. Большинство компиляторов расширяют указанные минимумы для одного или нескольких типов. От конкретной реализации компилятора также зависит, каким является тип char: знаковым или беззнаковым.

Тип	Типичный размер в байтах	Минимально допустимый диапазон значений			
char	1	от	-128	до	+127
unsigned char	1	от	0	до	255
signed char	1	от	-128	до	+127
int	2	от	-32768	до	32767
unsigned int	2	от	0	до	65535
signed int	2	от	-32768	до	32767
short int	2	от	-32768	до	32767
unsigned short int	2	от	0	до	65535
signed short int	2	от	-32768	до	32767
long int	4	от	-2147483648	до	2147483647
unsigned long int	4	от	0	до	4294967295
signed long int	4	от	-2147483648	до	2147483647
float	4	от	3.4e-38	до	3.4e+38
double	8	от	1.7e-308	до	1.7e+308
long double	10	от	3.4e-4932	до	1.1e+4932

2.6 КВАЛИФИКАТОР ТИПА const

Квалификатор типа const предоставляет дополнительную информацию о переменной, перед именем которой он находится. Объекты типа const не могут быть изменены программой в процессе ее выполнения. Кроме того, объект, адресуемый с помощью указателя, который определен как const, также не может быть модифицирован.

Компилятор может поместить переменные этого типа в память, предназначенную только для чтения (read-only memory - ROM). Переменная, определенная как const, получает свое значение при инициализации. Например, в результате выполнения строки const int zone = 10; будет создана целая переменная с именем zone и со значением 10, которое не может быть модифицировано программой. Тем не менее, эту переменную вполне можно использовать в выражениях других типов.

2.7 КОНСТАНТЫ

Константы, называемые также литералами, относятся к фиксированным значениям, которые не могут быть изменены программой. Константы могут иметь любой основной тип данных. Способ представления каждой константы зависит от ее типа. Символьные константы заключаются в одинарные кавычки. Например, ' a' и ' + ' являются символьными константами. Целочисленные константы задаются как числа без дробной части. Например, 360 и -180 это целочисленные константы. Вещественные константы содержат десятичную точку, за которой следует дробная часть числа, например 3.14159. Для вещественных констант можно также использовать экспоненциальное представление чисел, например 6.371e3 или 6.371E3. Такой записью представлено число $6.371*10^3$.

Существует два вещественных типа: float и double. Кроме того, существует несколько основных типов, которые образуются с помощью модификаторов типов. По умолчанию компилятор присваивает числовой константе совместимый и одновременно наименьший по занимаемой памяти тип данных. Единственным исключением из правила "наименьшего типа" являются вещественные (с плавающей точкой) константы, которым по умолчанию присваивается тип double.

Чтобы задать точный тип числовой константы, используйте соответствующий суффикс. Для вещественных типов действуют следующие суффиксы: если вещественное число завершить буквой F, оно будет обрабатываться с использованием типа float, а если буквой L, подразумевается тип long double. Для целых типов суффикс U означает использование типа unsigned, а суффикс L — тип long. Ниже приведены некоторые примеры представления констант с суфиксами.

Тип данных	Примеры констант
int	1, 31, 1852, -234
long int	35000L, -34L
unsigned int	3438U, 987U
float	57.3F, 4.34e-3F
double	123.23, 6371.110, -0.9876324
long double	1001.2L

2.8 ШЕСТНАДЦАТЕРИЧНЫЕ И ВОСЬМЕРИЧНЫЕ КОНСТАНТЫ

Иногда удобно вместо десятичной системы счисления использовать восьмеричную или шестнадцатеричную. В восьмеричной системе основанием служит число 8, а для выражения всех чисел используются цифры от 0 до 7. Число 10_8 в восьмеричной системе представляет число 8_{10} в десятичной системе счисления. Здесь основание системы отсчета записано подстрочным индексом. В программе, чтобы иметь возможность различать системы счисления числа в восьмеричной системе начинаются с нуля, например число 10_8 должно быть записано в виде 010.

Система счисления по основанию 16 называется шестнадцатеричной и использует цифры от 0 до 9 плюс буквы от A до F, означающие шестнадцатеричные "цифры" 10, 11,12,13,14 и 15. Например, шестнадцатеричное число 10 равно числу 16 в

десятичной системе. Поскольку эти две системы счисления (шестнадцатеричная и восьмеричная) используются в программах довольно часто, в языках C и C++ разрешено при желании задавать целые константы не в десятичной, а в шестнадцатеричной или восьмеричной системе. Шестнадцатеричная константа должна начинаться с префикса 0x (нуль и буква x) или 0X. Приведем два примера.

```
int  hex =  0x50;   //   80  в десятичной системе
int  oct  =  011;    //   9  в десятичной системе
```

2.9 СПЕЦИАЛЬНЫЕ УПРАВЛЯЮШИЕ КОНСТАНТЫ

Некоторые символы, например символ перехода на новую строку невозможно ввести в текст программы с клавиатуры, поскольку на клавиатуре такого знака просто нет. Символ перехода на новую строку кодируется специальным образом. Он состоит из двух знаков: наклонной четы и буквы n и записывается в виде \n, при этом данная последовательность рассматривается как один символ. В языках C и C++, помимо символа перехода на новую строку, используют и другие наборы управляющих констант. Приведем список этих констант.

Код	Значение
\b	Возврат на одну позицию
\f	Подача бланка
\n	Новая строка
\r	Возврат каретки
\t	Горизонтальная табуляция
\"	Двойная кавычка
\'	Одинарная кавычка (апостроф)
\ \	Обратная косая черта
\v	Вертикальная табуляция

Управляющие константы можно использовать там, где возможно использование обычных символов. Например, следующий оператор выполняет переход на новую строку, выполняет табуляцию (печатает пробелы), а затем печатает строку "Gauss coordinates".

```
cout << "\n\tGauss coordinates";
```

2.10 ЛОГИЧЕСКИЕ (БУЛЕВЫ) КОНСТАНТЫ

В языке C булевы константы не определены, ложь кодируется нулем, истина любым ненулевым числом, включая и отрицательные числа. В языке C++ специально определены две булевы константы true и false, с особым типом данных bool. Результатом вычисления выражений, в которых используются операторы сравнения и логические операторы, являются константы true и false.

2.11 СТРОКОВЫЕ КОНСТАНТЫ

Языки C и C++ поддерживают еще один встроенный тип данных, именуемый строковым. Строка - это набор символов, заключенных в двойные кавычки, например "Krasovskii’s ellipsoid". Не следует путать строки с символами. Символьная константа заключается в одинарные кавычки, например 'y'. Однако "y" - это уже строка, содержащая только одну букву. Строковые константы автоматически завершаются нулевым символом при компиляции. Кроме того, в языке C++ поддерживается класс string.

3. ОПРЕРАТОРЫ

В языках С и С++ предусмотрено большое количество операторов, которые можно разделить на следующие основные категории: арифметические, логические, побитовые (поразрядные). В языке С++, в отличие от С, существуют операторы ввода-вывода, а именно оператор вставки в поток << и оператор извлечения из потока >>.

Следует обратить внимание на то, что символами << и >> обозначаются и побитовые операторы сдвига влево и вправо. Различить эти две группы операторов можно по контексту.

3.1 АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ

Арифметическим оператором называют символ, который представляет определенной действие. Арифметические операторы являются бинарными, потому что они выполняют действия над двумя операндами. В языках С и С++ применяются семь арифметических операторов. Они представлены в таблице.

Оператор	Действие
-	Вычитание, унарный минус
+	Сложение
*	Умножение
/	Деление
%	Деление по модулю
--	Декремент
++	Инкремент

Операторы +, -, * и / выполняют сложение, вычитание, умножение и деление. Однако арифметическое деление в С и С++ имеет свою особенность. При делении целых дробная часть отбрасывается. Например, в результате деления ¾ будет 0, поскольку компилятор воспринимает эту операцию как деление целых чисел, числитель равен целому числу - трем и знаменатель четырем. Чтобы получить результат без отбрасывания дробной части, надо делимое или делитель представить вещественным числом, поставив например, после числа 3 точку.

Оператор % возвращает остаток от деления нацело (целочисленного деления). Выражение x % y дает остаток от деления x на y и, следовательно, равно нулю, когда x делится на y точно. Этот оператор нельзя применять к типам данных с плавающей точной. При целочисленном делении ½ результатом будет 0, а сам остаток, т.е. результат 1%2 равен 1.

Оператор инкремента (увеличения) ++ увеличивает переменную на 1, оператор декремента (уменьшения) -- уменьшает на 1. Операторы ++ и – могут записываются либо в префиксной форме то есть перед переменной ++*a* (--*a*), либо в постфиксной форме, после переменной *a*++ (*a*--). В обоих случаях происходит увеличении (уменьшение) *a* на 1. Эффект от применения префиксной и постфиксной форм получается в выражениях с присваиванием. Например, рассмотрим два фрагмента программы

```
a = 1;                    a = 1;
…                         …
x = ++a;                  x = a++;
```

В выражении вида x =++a переменная *a* вначале увеличивается, затем происходит ее присваивание переменной x. Другая последовательность действий выполняется в выражении x =a++, где переменная *a* увеличивается после того, как ее

значение было присвоено x . Таким образом, если в начале фрагмента кода a = 1, то после выполнения x = ++a переменные *a* =2, x =2, однако после выполнения x = a++ переменная *a* =2, но переменная x = 1.

Все рассмотренные выше арифметические операторы выполняются в порядке представленном в следующей таблице. Операторы одного уровня старшинства вычисляются слева направо.

Приоритет	Операторы
Наивысший	++ -- -(унарный минус)
	* / %
Низший	+ -

3.2 ОПЕРАТОР ПРИВЕДЕНИЯ ТИПА

Оператор приведения типа заставляет компилятор преобразовать один тип данных в другой. Как язык С, так и язык С++ поддерживают следующую форму записи операции приведения типа.

(тип) выражение

Здесь элемент тип означает желаемый тип данных.

Например, после выполнения следующей операции приведения типа результат деления заданных целых чисел будет иметь тип double.

```
double d;
d = (double) 10/3;
```

Это еще один способ избежать отбрасывания дробной части при делении.

3.3 ОПЕРАТОРЫ ОТНОШЕНИЙ И ЛОГИЧЕСКИЕ ОПЕРАТОРЫ

Операторы отношений и логические операторы используются для получения результатов в виде значений true/false ("истина"/"ложь"). В языках С и С++ любое ненулевое число оценивается как true. Нуль эквивалентен значению false. В языке С++ результат выполнения операторов отношений или логических операторов имеет тип bool, а в языке С - тип int (нуль или ненулевое целое). Ниже перечислены операторы отношений.

Оператор	Значение
>	Больше
>=	Больше или равно
<	Меньше
<=	Меньше или равно
==	Равно
!=	Не равно

Логические операторы применяются для объединения операторов сравнения в одно логическое выражение, которое можно, например, поместить в условие оператора if или в заголовок цикла. Далее следует список логических операторов.

Оператор	Значение
&&	И
||	ИЛИ
!	НЕ

Чтобы определить результат логической операции, например A&&B используется таблица истинности вида "A логическая операция B", которая представлена ниже

A	B	A&&B	A||B	!A
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

Унарная операция отрицания ! преобразует ненулевой или истинный операнд в 0, а нулевой или ложный операнд в 1.

Операторы отношений используются для сравнения, а логические операторы - для объединения двух значений или (в случае оператора !) для инвертирования значения. Приоритет этих операторов показан в следующей таблице.

Приоритет	Операторы
Наивысший	!
	> >= < <=
	== !=
	&&
Низший	||

В качестве примера рассмотрим, оператор if, который оценивает выражение в скобках как равное значению true и выводит на экран строку - Отметка точки меньше 100.

Пусть h = 90;

if(h < 100) cout << " Отметка точки меньше 100";

Однако в приведенном далее примере сообщение выведено не будет, поскольку для того, чтобы результат был равен значению true, оба операнда, связанные оператором &&, должны быть истинными (равными значению true).

Пусть h = 9; z = 9;

if(h < 10 && z > 10) cout << "Эта строка выведена не будет.";

3.4 ОПЕРАТОРЫ ПРИСВАИВАНИЯ

В языках C и C++ оператором присваивания служит одиночный знак (=) равенства. Простое присваивание записывается в виде, например height = 10. Присваивая одинаковое значение сразу нескольким переменным, можно записать цепочкой несколько присваиваний. Например, программная строка h1 = h2 = h3 = 10; присваивает значение 10 переменным h1, h2 и h3.

Языки C и C++ поддерживают "сокращенный" вариант операторов присваивания. Если полная версия оператора присваивания выражается в общей форме в виде *переменная = переменная оператор выражение*, то запись такого типа можно сократить до "укороченного" варианта: *переменная оператор = выражение*.

Например, операторы присваивания

length = length + 10; и

distance = distance /scale;

можно переписать в сокращенном виде:

length += 10; и

distance /= scale;

3.5 ОПЕРАТОР «ЗАПЯТАЯ»

Оператор «запятая» (иначе называется оператором последовательного вычисления) указывает на необходимость выполнения некоторой последовательности операций. Значение всего выражения, состоящего из списка выражений, разделенных запятыми, равно значению последнего выражения в этом списке.

Оператор "запятая" чаще всего используется в цикле for, например:

```
for(z = 10,  b = 20;  z < b;  z++,  b--)  {  ...  }
```

Здесь переменные z и b инициализируются и модифицируются с использованием списка выражений, разделенных запятыми.

3.6 ОПЕРАТОРЫ ВВОДА-ВЫВОДА

В языке C++ операторы << и >> используются для выполнения операций ввода-вывода. Если в выражении левым операндом является поток, то символы >> служат для обозначения оператора ввода, а символы << - оператора вывода. В языке C++ оператор >> называется *экстрактором* (extractor), поскольку он выделяет данные из входного потока. Оператор << можно назвать *оператором вставки* (inserter), поскольку он вставляет данные в выходной поток. Общие формы записи этих операторов имеют следующий вид.

входной_поток >> переменная

выходной_поток << выражение

Например, приведенный ниже фрагмент программы обеспечивает ввод значений двух целых переменных.

```
int  height1,  height2;
cin >>  height1 >> height2;
```

После выполнения следующего оператора на экране появится надпись Heights 10 , 20.

```
cout  <<  Heights"  <<  10  <<  ' , '  <<  4*5;
```

3.7 СВОДНАЯ ТАБЛИЦА ПРИОРИТЕТОВ ОПЕРАТОРОВ

В таблице показан приоритет выполнения всех операторов C и C++.

Приоритет	Операторы
Наивысший	( ) [ ] -> :: .
	! ~ ++ -- - * & sizeof new delete typeid
	Операторы приведения типа
	. * -> *
	* / %
	+ -
	<< >>
	< <= > >=
	== !=
	&
	^
	|
	&&
	||
	?:
	= += -= *= /= %= >>= <<= &= ^= !=
Низший	,

Все операторы, за исключением унарных операторов, операторов присваивания и оператора вопроса (?:) выполняются слева направо. Операторы одного уровня старшинства вычисляются слева направо.

4. КЛЮЧЕВЫЕ СЛОВА

Ключевыми словами языка программирования называется набор слов, формирующих синтаксис языка. Их нельзя использовать в качестве имен при определении переменных, функций, классов, поскольку они зарезервированы в компиляторе. В языке С определено 32 ключевых слова. Все ключевые слова состоят только из строчных букв.

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

В языке С++ содержатся все ключевые слова, определенные в языке С и дополнительные.

asm	inline	this
bool	reinterpret_cast	throw
catch	mutable	true
class	namespace	try
const_cast	new	typeid
delete	operator	typename
dynamic_cast	private	using
export	protected	virtual
explicit	public	wchar_t
false	static_cast	
friend	template	

Далее рассмотрены основные ключевые слова и их назначение, начиная с ключевых слов, которые задают порядок выполнения инструкций программы. Это if-else, switch, while, do-while и for, а также break и continue.

5. УПРАВЛЯЮЩИЕ КОНСТРУКЦИИ

В языках С и С++ существует несколько видов управляющих конструкций: условные операторы, операторы циклов и т. д.

5.1 if - else

Управляющая конструкция if - else позволяет менять направление выполняемых программой действий в зависимости от результата проверки условия. Общая форма записи конструкции if - else имеет следующий вид.

if (*условие*) {
блок_операторов_1 }
else {
блок_операторов_2
}

В качестве *условия* можно использовать любое выражение. Если это выражение в процессе выполнении программы становится истинным (любое значение, отличное от нуля), то будет выполнен *блок_операторов_1*; в противном случае будет выполнен

блок_операторов_2, если таковой существует. Оператор else необязателен и логическая конструкция часто используется в сокращенной форме в виде:

if (*условие*) { *блок_операторов* }

В этом случае, если *условие* ложно, то *блок_операторов* не выполняется и выполняется код, следующий за логической конструкцией.

В следующем фрагменте программы проверяется, больше ли значение переменной d числа 10, если это так, то выводится сообщение "d больше 10", иначе сообщение "d меньше или равен 10".

```
if(d > 10)
cout << "d больше 10";
else
cout << "d меньше или равен 10";
```

5.2 switch

Оператор switch - оператор многоканального ветвления. Он используется для организации работы программы по одному из нескольких направлений. Общая форма записи этого оператора имеет следующий вид.

witch {*выражение*) {
case *константа_1*: *последовательность_операторов_1*; break;
case *константа_2*: *последовательность_операторов_2*; break;
.
.
.
case *константа_N*: *последовательность_операторов_ N*; break;
default: default-*операторы*;
}

Каждая последовательность операторов может состоять из одного или нескольких операторов. Раздел default необязателен. Как выражение, так и константы, расположенные поле оператора case, должны иметь целочисленный тип.

Работа оператора switch заключается в сравнении *выражения* с константами. При обнаружении совпадения выполняется соответствующая последовательность операторов. Если в выполняемой последовательности операторов нет break, программа перейдет к выполнению операторов, относящихся к следующем операторе case. Такая ситуация часто называется «проваливанием» в следующий оператор case. Если не обнаружится никакое совпадение, то при наличии оператора default будет выполняться последовательность операторов, относящаяся к этой ветви. В противном случае (при отсутствии default) никакие действия выполнены не будут. В следующем примере переключатель switch используется для вызова разных функций и таким образом моделируется выбор команд меню.

```
switch(ch) {
case '0':  Exit();                                  break;
case '1':  TestOfTheodolite();                      break;
case '2':  MeasurementHorizontalAngles();           break;
case '3':  MeasurementVerticalAngles();             break;
default:   cout << "Ошибка, неизвестная команда!\n";
}
```

5.3 ЦИКЛЫ – while, do - while И for

Циклы используются для выполнения повторяющихся операций. В языках С и С++ имеются три оператора цикла: while, do-while и for.

5.4 while

Синтаксическая форма для цикла while выглядит следующий образом.

while{*условие*) {
тело цикла
}

Если *тело цикла* while состоит только из одного оператора, фигурные скобки можно опустить.

При выполнении цикла while сначала проверяется условие. Если условие дает ложный результат при первой же проверке, *тело цикла* не выполняется ни разу. Условие может быть любым выражением. Условие должно изменяться внутри тела цикла. Поэтому после выполнения некоторого количества циклов, условие становится ложным и выполнение цикла заканчивается.

Ниже приведен пример цикла while, который читает с клавиатуры 25 высот точек и сохраняет их в вещественном массиве.

```
double point_height[25];
int t = 0;
while( t < 25) {
cin >> point_height[t];
t++;
 }
```

5.5 do - while

Цикл do – while это разновидность цикла while. Его особенность состоит в том, что сначала происходит выполнение цикла, а затем проверка условия. Общая форма записи цикла do - while имеет такой вид.

do {
тело цикла
} while(*условие*);

Если при выполнении цикла повторяется только один оператор, то фигурные скобки можно опустить, хотя они делают программный код более понятным.

Цикл do - while гарантирует выполнение тела цикла как минимум один раз, поэтому часто такой цикл называют циклом с постусловием. Цикл while называют циклом с предусловием.

5.6 for

Цикл for используется в тех случаях, когда заранее известно количество повторяющихся операций. Общая форма записи представлена ниже.

for{*инициализация; условие; инкремент*)
{
блок_операторов
}

Цикл работает следующим образом. Сначала выполняется инициализация переменной счетчика цикла, т.е. выполняется выражение *инициализация*. Затем проверяется *условие*, если *условие* истинно, то выполняется тело цикла, после выполнения тела цикла происходит выполнение выражения *инкремент*, стоящего в качестве третьего параметра в заголовке цикла. Если же *условие* ложно, то тело цикла не выполняется и цикл завершается. Если тело цикла состоит только из одного оператора, фигурные скобки часто опускают.

Условие обычно представляет собой оператор отношения, сравнивающий значение переменной цикла с ее конечным значением, а инкремент увеличивает (или уменьшает) значение переменной цикла. Если условие дает ложное значение сразу после инициализации, тело цикла for ни разу не будет выполнено.

При выполнении следующего оператора цикла будет выведено десять раз с новой строки «Аэрофотоснимок» с возрастающим от 0 до 9 номером.

```
for(t=0;  t<10;  t++)  cout  <<  "Aerial Photograph " << t <<endl"
```

5.7 break

Ключевое слово break используется для выхода из циклов do, for и while в обход обычного условия выхода из цикла. Оно также используется для выхода из блока оператора switch.

Ниже показан пример использование оператора break в цикле do-while. В цикле вызывается функция getCoordinateY(), которая берет Y-координату точки, далее эта величина как-то преобразовывается функцией TransformCoordinate(y). Цикл заканчивается, если значение Y-координаты отрицательно.

```
do  {
y  =  getCoordinateY();
if (y <  0) break;
TransformCoordinate(y);
} while(!done);
```

В переключателе switch оператор break прерывает выполнение выбранной ветви case и управление передается на оператор, следующий за switch. Оператор break прекращает работу только тех циклов for, do, while и оператора switch, в которые он непосредственно вложен.

5.8 continue

Оператор continue используется для пропуска оставшихся операторов в теле цикла и передает управление на следующую итерацию, которая начинается с вычисления условного выражения. В нижеследующем примере цикл while будет читать, и обрабатывать X-координаты точек, пропуская точки с отрицательными X-координатами.

```
while(!done)  {
x  =  getCoordinateX();
if(x < 0)  continue;
process(x);
}
```

Обращение к функции process () не произойдет, если x < 0.

6. МАССИВЫ

Массивом называется набор данных одного типа, имеющих общее имя. Характерным признаком массива являются квадратные скобки, следующие за именем массива. В скобках находится индекс массива, с помощью которого можно непосредственно обратиться к элементу массива. Общая форма объявления одномерного массива имеет следующий вид.

тип Имя_массива[*размер*];

Здесь тип определяет тип данных для каждого элемента в массиве, а размер указывает количество элементов в массиве. Отсчет элементов массива, то есть индекс массива начинается с 0, а не с 1. Например, чтобы объявить целочисленный массив control_point из 100 элементов, запишите следующее.

```
int  control_point [100];
```

Эта запись создает массив, содержащий 100 элементов, номер первого элемента - 0, а последнего - 99. Поэтому, при выполнении следующего цикла в массив control_point загрузятся числа от 0 до 99.

```
for(n= 0;  n<100;  n + +)  control_point [n]  =  n ;
```

Многомерные массивы это массивы, содержащие более одного измерения. Каждое измерение имеет свой индекс, который записывается в своих квадратных

скобках. Размерность массива не ограничена в языках С и С++, но как правило используются одномерные и двумерные массивы. Двумерные массивы применяются для работы с математическими объектами – матрицами. Например, чтобы объявить массив целых размерностью 3 * 5, необходимо записать следующее.

```
int coordinates[3][5];
```

Массивы можно инициализировать с помощью списка инициализаторов, заключенного в фигурные скобки, как, например, показано ниже.

```
int point[5] = {1, 2, 3, 4, 5 };
int гамма[4][2]={ {7,45}, {-3,-25}, {5, 20}, {-4, -5} };
```

7. ФУНКЦИЯ main()

Функция main() – это главная функция программы, поэтому выполнение программы начинается именно с вызова функции main(). Программа не должна иметь более одной функции main(). По окончании работы функции main() выполнение программы завершается и управление возвращается операционной системе. Функция main(), подобно всем другим функциям, должна объявить тип возвращаемого значения, как правило, это тип целый. Можно использовать различные формы функции main(). Как для языка С, так и для языка С++ допустимы следующие варианты записи функции main():

```
int main()
int main(int argc, char *argv[])
```

Как видно из второй формы записи, функция main () поддерживает, по крайней мере, два параметра. Они называются argc и argv. Эти две переменные содержат количество аргументов командной строки и указатель на них соответственно. Параметр argc имеет целый тип, и его значение всегда будет не меньше числа 1, поскольку первым аргументом всегда является имя программы. Параметр argv должен быть объявлен как массив символьных указателей, в котором каждый элемент указывает на аргумент командной строки.

7.1 return

Если функция возвращает значение, она должна содержать выражение с оператором return. Оператор return обеспечивает передачу управления из функции обратно в вызывающую функцию и может быть использован также для передачи некоторого результата. Он имеет следующие две формы записи.

```
return;
return значение;
```

Оператор return может находиться в любой части тела функции, но обычно его заканчивает. В языке С++ форма оператора return, которая не задает возвращаемого значения, должна использоваться только в void-функциях.

Следующая функция возвращает произведение двух своих аргументов.

```
int multiply(int a, int b)
{
return a*b;
}
```

Следует иметь в виду, что как только компилятор обнаружит оператор return, сразу же будет выполнен возврат из функции и любой код, который может присутствовать в функции после оператора return, будет опущен. Кроме того, функция может содержать более одного оператора return.

8. ПРОСТРАНСТВА ИМЕН

В языке C++ можно создать локальную область видимости, используя ключевое слово namespace. Пространство имен определяет некоторую декларативную область. Ее назначение — ограничить действие имен. Общая форма задания пространства имен имеет следующий вид:

namespace *имя* {
// …
}

Здесь имя означает имя пространства имен. Приведем пример.

```
namespace Theodolite {
int count;
 }
```

Этим объявлением создается пространство имен Theodolite, а внутри него объявляется переменная count.

На имена, объявленные внутри некоторого пространства имен, могут напрямую ссылаться другие операторы внутри того же пространства имен. Вне своего пространства имен к именам можно получить доступ двумя путями. Во-первых, можно использовать оператор разрешения области видимости. Например, с учетом приведенного выше объявления Theodolite следующий оператор вполне правомочен.

```
Theodolite::count = 10;
```

Во-вторых, можно использовать оператор using, который привносит заданное имя или пространство имен в текущую область видимости, как в примере, приведенном ниже.

```
using namespace Theodolite;
count = 100;
```

В данном случае к переменной count можно обращаться напрямую, поскольку она теперь относится к текущей области видимости.

Элементы, объявленные в библиотеке C++, относятся к пространству имен std.

9. СТАНДАРТНЫЕ БИБЛИОТЕКИ С И С++

В языках C/C++ нет операторов, обеспечивающих ввод-вывод, обрабатывающих строки, выполняющих различные математические вычисления. Все эти операции выполняются за счет использования набора библиотечных функций, поддерживаемых компилятором. Существует два основных вида библиотек: библиотека C-функций, которая поддерживается всеми компиляторами C и C++, и библиотека классов C++ для языка C++. Чтобы программа могла использовать какую-нибудь библиотеку функций, она должна иметь соответствующий заголовок. Вообще-то, под заголовками понимают заголовочные файлы, но на самом деле это необязательно должны быть именно файлы. Однако для всех практических нужд стандартные C-заголовки содержатся в файлах, которые соответствуют их именам. В таблице перечислены стандартные заголовки.

Заголовок	Что поддерживает
<ctype. h>	Обработка символов
<math. h>	Различные определения, используемые библиотекой math
<stdio.h>	Ввод-вывод файлов
<string. h>	Функции обработки строк
<time. h>	Функции системного времени и даты

В современной спецификации для языка C++ заголовки указываются с использованием стандартных имен заголовков, которые не имеют расширения . h (т.е.

заголовки C++ не означают имена файлов). Это просто стандартные идентификаторы. В таблице ниже приведены C++ -заголовки.

Заголовок	Что поддерживает
<complex>	Комплексные числа
<exception>	Обработка исключительных ситуаций
<iomanip>	Манипуляторы ввода-вывода
<iostream>	Обработка входных потоков
<string>	Стандартный класс string
<vector>	Векторы (динамические массивы)

В стандартном C++ вся информация, связанная со стандартной библиотекой, определена в пространстве имен std. Следовательно, для получения прямого доступа к этим элементам после включения нужного заголовка необходимо использовать следующий оператор using.

using namespace std;

В качестве альтернативного варианта, чтобы не вносить целую библиотеку в глобальное пространство имен, каждый библиотечный идентификатор можно квалифицировать с помощью обозначения std::, например std::cout. Однако в этом случае каждое имя будет выглядеть весьма громоздко.

Старые версии компиляторов C++, не поддерживают C++ -заголовки нового стиля и команду namespace. В этом случае необходимо использовать заголовки более старого традиционного стиля. Для этого к именам заголовков нужно добавить расширение .h, после чего они будут напоминать C-заголовки. Например, так выглядит включение заголовка <iostream> при использовании традиционного подхода.

#include <iostream.h>

При использовании заголовка в традиционном стиле все имена, определенные этим заголовком, размещаются в глобальном пространстве имен, а не в пространстве имен std. Следовательно, в этом случае оператор using не требуется.

10. ПРЕПРОЦЕССОР

Языки C и C++ включают ряд директив препроцессора, которые используются для выдачи инструкций компилятору.

Рассмотрим следующие директивы:

10.1 #define

Директива #define используется для выполнения макроподстановок одного текстового фрагмента другим по всему файлу, в который включена эта директива. Общая форма записи директивы #define имеет следующий вид.

#define *имя_макроса последовательность_символов*

После включения этой директивы каждое вхождение текстового фрагмента, определенное как *имя_макроса*, заменяется заданным элементом *последовательность_символов*. Обратите внимание на то, что в этом операторе нет точки с запятой. Кроме того, заданная *последовательность_символов* завершается только символом конца строки.

Например, если вы хотите использовать значение 1 для слова TRUE и значение 0 для слова FALSE, объявите эти два макроса с помощью отдельных директив #define.

#define TRUE 1

#define FALSE 0

Данные операторы вынудят компилятор подставлять 1 или 0 каждый раз, когда встретится слово TRUE или FALSE соответственно.

10.2 #include

Директива #include заставляет компилятор включить в программу файл с заданным именем. Для записи этой директивы используются следующие общие формы.

#include "*имя_файла*"

#include <*имя_файла*>

Имя исходного файла, подлежащего включению, должно быть заключено в двойные кавычки или угловые скобки. Например, оператор

#include “FieldBook.h”

заставит компилятор прочитать и включить файл FieldBook.h.

Если имя файла заключено в угловые скобки, то файл, находится в некотором каталоге (include), специально выделенном для заголовочных файлов. Если же имя файла заключено в кавычки, поиск файла выполняется другим способом. Как правило, поиск файла выполняется в текущем рабочем каталоге. Если заданный файл не найден, поиск повторяется с использованием первого способа (как если бы имя файла было заключено в угловые скобки). Чтобы ознакомиться с подробностями, связанными с различной обработкой директивы #include в случае использования угловых скобок и двойных кавычек, обратитесь к руководству пользователя, прилагаемому к вашему компилятору.

Помимо включения файлов, C/C++ - программы используют директиву #include для включения заголовков. В языках C и C++ определен набор стандартных заголовков, которые предоставляют информацию, необходимую для различных библиотек (т.е. под заголовком может подразумеваться файл, но это совсем необязательно). Однако на практике C -заголовки почти всегда являются файлами. Но для языка C++ ситуация совсем иная. Все имена C++ -заголовков представляют собой стандартные идентификаторы, которые компилятор может преобразовать в имена файлов или обработать каким-либо другим способом. Поскольку C++ -заголовки не являются именами файлов, они не имеют расширения .h. Например, чтобы включить заголовочную информацию для системы ввода-вывода, используйте следующий оператор.

#include <iostream>

Здесь <iostream> - стандартный заголовок для классов ввода-вывода.

11. ЧАСТО ИСПОЛЬЗУЕМЫЕ МАТЕМАТИЧЕСКИЕ ФУНКЦИИ

Функция	Запись на C++
sin x	sin(x)
cos x	cos(x)
tg x	tan(x)
arcsin x	asin(x)
arccos x	acos(x)
arctg x	atan(x)
$\sqrt{x}$	sqrt(x)
x^y	pow(x,y)
e^x	exp(x)
ln x	log(x)
lg x	log10(x)
|x|	fabs(x)
остаток от деления x на y	fmod(x,y)

Заметим, что аргументом стандартных тригонометрических функций sin(), cos() и tan() должен быть угол, выраженный в радианах. Число π - стандартная константа из заголовочного файла <math.h>, включить константу в код программы можно по имени

M_PI, имеется также константа $\frac{\pi}{2}$ - M_PI_2 и константа $\frac{\pi}{4}$ - M_PI_4 соответственно. В библиотеке стандартных математических функций имеются обратные тригонометрические функции: арксинус - asin(), арккосинус - acos(), арктангенс - atan(), нет функции котангенса и арккотангенса.

12. ОПРЕДЕЛЕНИЕ И ВЫЗОВ ФУНКЦИИ

В определении функции используется следующий синтаксис:

```
тип   имя_функции(список_аргументов)    //аргументы с указанием типа
                                         //каждого из них отделяются
                                         //друг от друга запятой
{
      Тело  функции                      //последовательность инструкций
                                         //после каждой инструкции
                                         //ставится точка с запятой (;)
}
```

После определения функции к ней можно обращаться в теле любой другой функции. Аргументы в определении функции являются формальными. Фактическое значение они приобретают только при обращении к функции. В литературе это действие еще называют вызовом функции. При вызове функции необходимо соблюдать соответствие между фактическими и формальными аргументами по типу, количеству и порядку следования.
Рассмотрим эти ситуации на следующем примере:

```
// ВАРИАНТ  1
//
// определим некоторую функцию
// используем ее для нахождения дискриминанта
// квадратного уравнения ax^2 + bx + c = 0

  float  F(float  a0 , float  a1 , float  a2)
         {
          float  d;
                 d=a1*a1-4*a0*a2;
                 return  d;
         }
 void   main()
 {
  int  a,b,c;
  float  dis;
  clrscr();
  cin>>a;
  cin>>b;
  cin>>c;
  dis=F(a,b,c);    //здесь будет выдано сообщение об ошибке несоответствие
  cout<<dis;       //по типу (в определении функции F a0,a1,a2 - объявлены типа
                   // float), поэтому при вызове необходимо, чтобы
                   // соответствующие фактические аргументы имели этот же тип
```

```
                    //программа выполняться не будет
  }

// ВАРИАНТ   2
//
//  используется  функция   F(a0,a1,a2)

//
  void   main()
  {
   float   a,b,c,dis;
   ……
   ……
   ……
   ……
   dis=F(a,b);       //будет выдано сообщение об ошибке несоответствие по
    …….. //количеству в определении функции 3 аргумента, // в вызове 2
                    //программа выполняться не будет
  }


//ВАРИАНТ   3
//
//  используется   функция   F(a0,a1,a2)
//
  void   main()
  {
   float   a,b,c,dis;
   ……
   ……
   ……
   ……
   dis=F(c,a,b);      //здесь несоответствие между формальными и фактическими
                    // аргументами по порядку следования
                    //при вызове F(c,a,b) значение  dis   определится как
                    // a2  -  4cb, такого рода ошибка при компиляции не
                    //обнаруживается
                    //программа будет выполнена с неверным результатом
   }
```

Учитывая изложенное, получаем

```
  void   main()
  {
   float   a,b,c,dis;
   clrscr();
   cin>>a;
   cin>>b;
   cin>>c;
   dis=F(a,b,c);
   cout<<dis;
  }    //программа будет выполнена правильно
```

13. УКАЗАТЕЛИ, ИХ ОБЪЯВЛЕНИЕ И ИСПОЛЬЗОВАНИЕ В ФУНКЦИЯХ

До сих пор мы рассматривали переменные, для работы с которыми в описательной части программы указывали их имя и тип. Компилятор превращал имя в адрес, а типом определялось число байтов, отводимое под эту переменную. Такой способ организации обращения к переменной называется прямой адресацией. При косвенной адресации переменная будет располагаться в ячейке, адрес которой находится в другой ячейке. Этот способ адресации используется для объявления переменных типа указатель.

Синтаксис объявления указателя следующий:

тип **имя_ переменной*;

Например, чтобы объявить p указателем для хранения адреса действительной переменной необходимо, чтобы и сама p была переменной действительной, т. е. написать

```
float   *p;
```

Аналогично и для других базовых типов.

13.1 ОПЕРАТОРЫ ДЛЯ РАБОТЫ С УКАЗАТЕЛЯМИ

* - оператор разыменования указателя,

& - оператор получения адреса,

Приоритет операторов * и & выше чем приоритет арифметических операторов (за исключением унарного минуса). Рассмотрим их работу на следующем примере:

```
float   *p;          //  здесь указатель  p  указывает на переменную типа  float
                     //  значение  p  -  неопределенное, перед использованием
                     //  указатель необходимо инициализировать, т.е. загрузить в  p
                     //  адрес некоторой переменной типа  float,  инициализация p
                     //  производится оператором получения адреса  (&)
float    t;
float    s=3.14;
         p=&s;       // здесь оператор присваивания записывает в указатель p
                     // адрес переменной  s
                     // по этому адресу теперь находится ячейка памяти, в которой
                     // хранится переменная   s
         t=*p;       // присваиваем  t  значение той переменной, адрес  которой
                     // загружен  в  p
                     // p  указывает  на  s  -  переменную  типа  float , ее значение
                     // равно 3.14,  теперь  и  t=3.14

         t=s;        // тот же результат
         s=2*t;      //              s=6.28
        *p =2**p;    // считываем значение  s,  на которую указывает  p
                     // умножаем на 2 и полученное значение присваиваем
                     // переменной, адрес которой находится в  p  ---  адрес  s
                     //              s=12.56
         p=&t;       // производим  повторную  инициализацию  указателя  p
                     // теперь  p  указывает  на  t  -  переменную  типа  float ,
```

```
                // сейчас   ее   значение -  3.14
s=2**p;    // теперь   s=6.28
                // т.е. работа с указателем сводится к работе с той
                // переменной, адрес которой предварительно загружен
                // операцией получения адреса (&)
cout<<*p;    // вывести на экран значение переменной, адрес которой
                 // находится в  p. Сейчас это значение  t = 3.14
cin>>*p;      // ввести с клавиатуры новое значение   t
```

13.2 УКАЗАТЕЛИ И МАССИВЫ

В C/C++ существуют два способа обращения к элементам массива:

1) индексация
2) адресная арифметика

Индексация массива нагляднее, адресная арифметика быстрее и чаще используется при обработке данных в режиме реального времени. Вероятность ошибки при написании программы здесь выше. Рассмотрим два примера. В первом варианте используется индексация, а во втором - адресная арифметика.

```
int   a[10] , *n ;
      n = &a[0] ;
```

Здесь указателю n присвоен адрес первого элемента массива a. Чтобы получить доступ к пятому элементу этого массива, следует выполнить один из двух операторов:

a[4] или *(n + 4)

Оба оператора вернут значение пятого элемента массива a. Индексация массива начинается с нуля, поэтому индекс пятого элемента массива a равен 4.

Рассмотрим пример использование указателя для обмена данными между вызываемой и вызывающей функциями. Ниже приведен полный текст программы нахождения суммы корней квадратного уравнения: $ax^2+bx+c=0$

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>
float dis(float b,float a,float c)        //определение функции вычисления
                                          //дискриминанта квадратного уравнения
{
     return b*b-4*a*c;
}
void  mess()                              //назначение данной функции – вывод
                                          //сообщения об отсутствии
                                          //решения и выход в редактор
{cout<<"\nnet korney";
getch();
exit(0);
}
void  korni(float b,float a,float c,float *x1,float *x2)  //назначение функции korni –
                                                          //вычисление корней
                                                          //квадратного уравнения и
```

```
                                                    //передача их значений в
                                                    //вызывающую функцию
{
cout<<"\n dis="<<dis(b,a,c);
if(dis(b,a,c)<0) mess();
     *x1=(-b+sqrt(dis(b,a,c)))/(2*a);  //найденное значение корня будет присвоено
                                       //переменной, фактический адрес которой
                                       //будет загружен в указатель x1, при
                                       //обращении к функции korni (при вызове
                                       //функции korni)
                                       //все вышесказанное относится и к указателю
                                       //x2
     *x2=(-b-sqrt(dis(b,a,c)))/(2*a);

}
 void main()

 { float a,b,c;
   float  x1,x2,s;
clrscr();                              //очистка экрана
cout<<"\n a=";cin>>a;                  //ввод коэффициентов a, b, c
cout<<"\n b=";cin>>b;
cout<<"\n c=";cin>>c;
     korni(b,a,c,&x1,&x2);             //вызов функции найденные korni
                                       //значения будут присвоены тем переменным,
                                       //адреса, которых указываются при вызове
cout<<"\n x1="<<x1;
cout<<"\n x2="<<x2;
s=x1+x2;                               //находим сумму корней и выводим ее
                                       //значение на экран
      cout<<"\n s="<<s;
getch();
}
```

При адресации данных массива указывается его начальный адрес. После отладки программы, необходимо создать библиотеку функций. Для этого создается новый файл с именем “bibl.cpp”, в который необходимо скопировать эти функции:

```
float dis(float b,float a,float c)
{
     return b*b-4*a*c;
}
void  mess()
{cout<<"\nnet korney";
getch();
exit(0);
}
void  korni(float b,float a,float c,float *x1,float *x2)
{
 cout<<"\n dis="<<dis(b,a,c);
          getch();
if(dis(b,a,c)<0) mess();
     *x1=(-b+sqrt(dis(b,a,c)))/(2*a);
```

```
    *x2=(-b-sqrt(dis(b,a,c)))/(2*a);
}
```

Затем, в исходной программе убираются скопированные функции, и подключается созданная библиотека:

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>
#include"e:\bibl.cpp"  //при подключении библиотеки, необходимо указать ее
                       //адрес. В данном случае, созданная библиотека находится
                       //на внешнем носителе e:\
 void main()
 { float a,b,c;
   float  x1,x2,s;
clrscr();
cout<<"\n a=";cin>>a;
cout<<"\n b=";cin>>b;
cout<<"\n c=";cin>>c;
     korni(b,a,c,&x1,&x2);
     cout<<"\n x1="<<x1;
     cout<<"\n x2="<<x2;
s=x1+x2;
      cout<<"\n s="<<s;
}
```

Пример 2. Рассмотрим следующую программу. С клавиатуры вводится целая матрица порядка 3x4, в которой все отрицательные элементы необходимо заменить нулями. В этой программе предварительно определим и используем следующие функции:

- Ввод данных матрицы (функция vvod)
- Обработка введенных данных (функция obr)
- Вывод результата (функция vivod)

Во всех функциях будет использоваться промежуточная матрица A, что позволит применять более наглядный способ обращения к ее элементам - индексацию.

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void vvod(int *p)              //при обращении к этой функции в указатель будет
                               //загружен начальный адрес вводимой матрицы
{
```

/* далее i – номер строки вводимой матрицы; j – номер ее столбца; x,y – текущие координаты курсора на экране монитора; a, b – координаты начального положения курсора; d – показатель, зависящий от количества знаков в отображаемом на экране элементе матрицы */

```
  int i,j,x,y,k;
  int a,b,d;
```

```
 int A[3][4];                    // A[3][4] – промежуточная матрица
 gotoxy(1,2);
 cout<<"a=";cin>>a;
 cout<<"b=";cin>>b;
 cout<<"d=";cin>>d;
 clrscr();
 x=a;y=b;
 for(i=0;i<=2;i++)
 { gotoxy(x,y);
 for(j=0;j<=3;j++)               // ввод данных в промежуточную матрицу
 { cin>>A[i][j]; x=x+d; gotoxy(x,y);}
 x=a; y=y+1;
 }
 k=0;
```

/* копирование данных из промежуточной матрицы в матрицу, начальный адрес которой будет присвоен указателю p, при обращении к функции ввода */

```
 for(i=0;i<=2;i++)
 for(j=0;j<=3;j++)

 { *(p+k)=A[i][j]; k=k+1;}
}
void obr(int *f)                 // обработка данных матрицы
{
 int i,j,k,A[3][4];              // A[3][4] – промежуточная матрица
 k=0;
 for(i=0;i<=2;i++)               // копирование данных в промежуточную
                                 // матрицу A из обрабатываемой матрицы,
                                 // начальный адрес, которой будет присвоен
                                 // указателю f, при обращении к функции obr
 for(j=0;j<=3;j++)
  {  A[i][j]=*(f+k);
     k=k+1;}
 for(i=0;i<=2;i++)               // замена отрицательных элементов промежуточной
                                 // матрицы A нулями
 for(j=0;j<=3;j++)
 if(A[i][j]<0) A[i][j]=0;
 k=0;
 for(i=0;i<=2;i++)               // копирование обработанной промежуточной
                                 // матрицы A в исходную матрицу
 for(j=0;j<=3;j++)
 { *(f+k)=A[i][j]; k=k+1;}
}
void vivod(int *f)               // функция вывода матрицы, начальный адрес
                                 //которой будет загружаться в f, при обращении к
                                 // этой функции
{
 int i,j,x,y,A[3][4],k;
 int a,b,d;
 gotoxy(1,2);
 cout<<"a=";cin>>a;
 cout<<"b=";cin>>b;
```

```
 cout<<"d=";cin>>d;
 k=0;
 for(i=0;i<=2;i++)
 for(j=0;j<=3;j++)
 { A[i][j]=*(f+k); k=k+1;}     // копирование выводимой матрицы в
                              // промежуточную матрицу A
 x=a;y=b;
 for(i=0;i<=2;i++)            // вывод матрицы A на экран
 { gotoxy(x,y);
 for(j=0;j<=3;j++)
 {cout<<A[i][j]; x=x+d; gotoxy(x,y);}
 x=a;y=y+1;}
}
void   main()                 // непосредственно сама программа, которые
                              // использует определенные выше функции
{
 int b[3][4];                 // b – это и есть обрабатываемая матрица
 clrscr();
 vvod(&b[0][0]);              // при обращении к функции ввода указатель p
                              // принимает значение начального адреса матрицы
                              // b[0][0]
 obr(&b[0][0]);               // аналогично при обращении к функциям obr и vivod
 vivod(&b[0][0]);
 getch();
}
```

После отладки программы, помещаем функции vvod, vivod и obr в библиотеку bibl2.cpp (будем считать, что она находится на носителе e:\), подключаем ее, и после чего программа приобретает следующий вид:

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
#include"e:\bibl2.cpp"
void   main()
{
 int b[3][4];
 clrscr();
 vvod(&b[0][0]);
 obr(&b[0][0]);
 vivod(&b[0][0]);
 getch();
}
```

14. ЗАДАЧИ ДЛЯ САМОСТОЯТЕЛЬНОГО РЕШЕНИЯ

Целью этих задач являются упражнения по составлению и использованию функций, определяемых пользователем. После проверки отлаженные функции необходимо убрать из программы и поместить их в библиотечный файл. Затем подключить созданную библиотеку к программе, запустить ее и убедиться в правильности работы.

1. Треугольник задан координатами своих вершин. Найти его периметр. Предварительно определить функцию для вычисления длины стороны.
2. Найти значение абсолютной величины x. Предварительно определить функцию для вычисления абсолютной величины.
3. Найти сумму высот в треугольнике, заданном координатами своих вершин. Предварительно определить функцию высоты.
4. Найти угол между двумя векторами на плоскости. Предварительно определить функцию для нахождения скалярного произведения двух векторов. Координаты векторов заданы.
5. На двух векторах как на сторонах построен параллелограмм. Координаты векторов заданы. Найти диагонали. Предварительно определить функцию скалярного произведения двух векторов.
6. Найти площадь выпуклого пятиугольника, заданного координатами своих вершин. Предварительно определить функцию длины стороны и использовать ее для определения функции площади треугольника, которую затем применить для пятиугольника.
7. Найти значение определенного интеграла методом трапеций с точностью eps. Определить подынтегральную функцию F(x) и функцию вычисления интеграла. Значение eps, a, b ввести с клавиатуры. Распечатать полученные значения интеграла и числа итераций для достижения точности eps .
8. Найти значение определенного интеграла методом касательных с точностью eps. Далее порядок работы повторяет порядок предыдущей задачи.
9. Найти корни уравнения F(x)=0 методом половинного деления. Для решения задачи создать следующие функции:
   a) функция, определяющая F(x);
   b) функция, выполняющая отделение корней;
   c) функция, выполняющая нахождение корней с требуемой точностью eps (здесь же подсчитывается необходимое для этого число итераций);

Примечание: в определении функций b) и c) использовать указатели.

В задачах 10 -13 определить и использовать функцию ввода исходных данных, применяя указатели.

10. Найти сумму площадей двух треугольников, заданных координатами своих вершин. Определить и использовать функцию для вычисления длины стороны треугольника.
11. Найти сумму медиан треугольников A1B1C1 и A2B2C2, заданных координатами своих вершин. Определить и использовать функцию высоты.
12. Найти сумму радиусов окружностей, вписанных в треугольники A1B1C1 и A2B2C2. Координаты вершин заданы. Предварительно определить функцию радиуса.
13. Найти сумму радиусов окружностей, описанных около треугольников A1B1C1 и A2B2C2. Предварительно определить функцию радиуса. Координаты вершин заданы.

В задачах с матрицами предварительно определить ввод, вывод т обработку как функции, а для обмена данными использовать указатели.

14. Ввести с клавиатуры и сложить две матрицы порядка 2x3. Результат вывести на экран.
15. Ввести матрицу порядка 3x4. Поменять местами первый и последний столбцы.
16. Ввести матрицу порядка 3x4. Поменять местами первую и последнюю строчки.
17. Ввести матрицу порядка 4x4. Поменять местами наибольший и наименьший элементы.

18. Ввести и транспонировать матрицу порядка 3x4.
19. Ввести матрицу порядка 4x4. Привести к треугольному виду и найти ее определитель.
20. Ввести с клавиатуры матрицы A[2][3] и B[3][4]. Найти их произведение.

15. СПИСОК ЛИТЕРАТУРЫ

1. Керниган Б., Ритчи Д. Язык программирования Си, 3-е изд. СПб.: «Невский Диалект», 2001, 352с.
2. Либерти Дж. Освой самостоятельно C++ за 21 день. М.: Изд. Дом «Вильямс», 2001, 816с.
3. Подбельский В.В., Фомин С.С. Программирование на языке Си. М.: Финансы и статистика, 2001, 600с.
4. Подбельский В.В. Язык Си++. М.: Финансы и статистика, 2003, 560с.
5. Джамса К. Учимся программировать на языке C++. М.: Мир,1999, 320с.
6. Дейтел Х.М., Дейтел П.Дж. Как программировать на Си.3-е изд. М.: Бином-Пресс, 2002, 1168с.
7. Шилдт Г. Самоучитель C++. СПб.: BHV –Санкт-Петербург, 1997, 511с.

16. СОДЕРЖАНИЕ