СПРАВОЧНЫЙ МАТЕРИАЛ

Содержание

Таблица 1. Базовые типы данных языка C++ 2
Таблица 2. Управляющие символы для потока вывода 3
Таблица 3. Операторы C++ 3
Таблица 4. Приоритеты операций и операторов 6
Таблица 5а. Математические функции из библиотеки `cmath` 8
Таблица 5б. Математические функции из заголовочного файла `cstdlib` 9
6. Ввод-вывод C++ 10
Таблица 6а. Флаги форматного вывода с примерами их использования 10
Таблица 6б. Манипуляторы форматного вывода с примерами их использования 11
7. Функция форматированного ввода/вывода 13
Таблица7а Спецификаторы формата вывода 13
Таблица7б Спецификаторы формата ввода 15
8. Описание массивов в программе на C++ 16

Таблица 1. Базовые типы данных языка C++

тип	байт	Диапазон принимаемых значений
целочисленный (логический) тип данных		
bool	1	true и false | 0 до 255 (от 0 до 2^8)
целочисленный (символьный) тип данных		
*signed char	1	-128 до 127 (от -2^7 до $+2^7-1$)
char	1	0 до 255 (от 0 до 2^8-1)
целочисленные типы данных		
*signed short int	2	-32 768 до 32 767 (от -2^{15} до $+2^{15}-1$)
unsigned short int	2	0 до 65 535 (от 0 до $+2^{16}-1$)
int	4	-2 147 483 648 до 2 147 483 647 (от -2^{31} до $+2^{31}-1$)
unsigned int	4	0 до 4 294 967 295 (от 0 до $+2^{32}-1$)
long long	8	от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 (-2^{63} до $+2^{63}-1$)
unsigned long long		от 0 до $2^{64}-1$ (до 18 446 744 073 709 551 615)
типы данных с плавающей точкой		
float	4	от 3.4E-38 до 3.4E+38
double	8	от 1.7E-308 до 1.7E+308
long doble	10	от 3.4E-4932 до 3.4E+4932

* по умолчанию все целочисленные типы считаются знаковыми, т.е. спецификатор `signed` можно опускать.

Таблица 2. Управляющие символы для потока вывода

Управляющие символы (или как их ещё называют — escape-последовательность) — символы которые выталкиваются в поток вывода, с целью форматирования вывода или печати некоторых управляющих знаков C++.

Символ	Описание
\r	возврат каретки в начало строки
\n	новая строка
\t	горизонтальная табуляция
\v	вертикальная табуляция
\»	двойные кавычки
\’	апостроф
\\	обратный слеш
\0	нулевой символ
\?	знак вопроса
\a	сигнал бипера (спикера) компьютера

Все управляющие символы, при использовании, обрамляются двойными кавычками, если необходимо вывести какое-то сообщение, то управляющие символы можно записывать сразу в сообщении, в любом его месте.

Таблица 3. Операторы C++

a. Арифметические операторы			
Операция (выражение)	**Оператор**	**Синтаксис выражения**	**Краткое описание/примечание**
Присваивание	=	a = b	
Сложение	+	a + b	
Вычитание	-	a - b	
Умножение	*	a * b	
Деление	/	a / b	для целых чисел работает как *целочисленное* деление:

			$x = 7/3 = 2$, но $x = 7./3 = 2.3333\ldots$
Операция модуль (остаток от деления *целых* чисел)		**%**	a **%** b	$x = 7\%3 = 1$
b. Унарные (одноместные) операторы				
Унарный плюс		+	+a	**+1**
Унарный минус		-	-a	**-3.**
*Инкремент	префиксный	++	++a	$a = 9;\ ++a = 10$
	постфиксный	++	a++	$a = 2;\ a++ = 3$
*Декремент	префиксный	--	--a	$a = 9;\ --a = 8$
	постфиксный	--	a--	$a = 2;\ a-- = 1$
Преобразование типа		**(тип) a**	(int)a | int(a)	$a = 2.1; b = (\text{int})a = 2;$

* При использовании операции *инкремента в префиксной* форме значение переменной, сначала, увеличивается на 1, а затем используется в выражении. При использовании операции *инкремента в постфиксной* форме значение переменной сначала используется в выражении, а потом увеличивается на 1. Аналогично обе формы работают и для декремента. Пример:

Пусть *b* = 7. Тогда 10+ ++ *b* равно 18, а *b* = 8.
Пусть *b* = 7. Тогда 10+ *b*-- равно 17, а *b* = 6.

c. Операторы сравнения			
Операция (выражение)	**Оператор**	**Синтаксис выражения**	**Краткое описание/примечание**
Равенство	==	a == b	$a = 1; b = 2; c = 1;$ $a == b\ ?\ false;\ a == c\ ?\ true$
Неравенство	!=	a != b	$a = 1; b = 2; c = 1;$ $a\ != b\ ?\ true;\ a\ != c\ ?\ false$
Больше	>	a > b	
Меньше	<	a < b	
Больше или равно	>=	a >= b	
Меньше или равно	<=	a <= b	

d. Логические операторы			
Операция (выражение)	**Оператор**	**Синтаксис выражения**	**Краткое описание/примечание**
Логическое отрицание, НЕ	**!**	**!**a	Не a
Логическое умножение, И	**&&**	a **&&** b	a И b
Логическое сложение, ИЛИ	||	a || b	a ИЛИ b

e. Составные присваивания			
Операция (выражение)	**Оператор**	**Синтаксис выражения**	**Значение**
Сложение, совмещённое с присваиванием	+=	a += b	a = a + b
Вычитание, совмещённое с присваиванием	-=	a -= b	a = a - b
Умножение, совмещённое с присваиванием	*=	a *= b	a = a * b
Деление, совмещённое с присваиванием	/=	a /= b	a = a / b
Вычисление остатка от деления, совмещённое с присваиванием	**%=**	a **%=** b	a = a **%** b

f. Другие операторы		
Оператор	**Синтаксис**	

Тернарная условная операция	a **?** b **:** c	a=1,b=3,c=2;
		z=a>b?a+b:c+b;
		z=5

Оператор расширения области видимости	a**::**b	
Приведение типа	(**тип**) a | **тип**(a)	float x,y,z; x=(int)z/int(y);
`sizeof` (аргумент) определение размера аргумента в байтах	**sizeof**(a) | **sizeof**(*type*)	int b[5]; cout<<sizeof(int)<<'\t'<<sizeof(b); будет напечатано: 4 20

Выделение памяти	**new** *type*	
Выделение памяти для массива	**new** *type***[**n**]**	
Освобождение памяти	**delete** a	
Освобождение памяти, занятой массивом	**delete[]** a	\

Таблица 4. Приоритеты операций и операторов

В данной таблице указаны приоритеты операторов и их ассоциативность. Операторы, указанные в таблице выше (раньше), имеют более высокий приоритет (приоритет вычисления). При рассмотрении выражения, операторы, имеющие более высокий приоритет, будут вычислены раньше операторов с низким приоритетом. Если несколько операторов указаны в одной ячейке, то они имеют одинаковый приоритет и вычисляются в последовательности, задаваемой ассоциативностью.

Приоритет	Оператор	Описание	Ассоциативность
1 Наивысший	::	Раскрытие области видимости	Нет
2	++	Постфиксный инкремент	Слева направо
	--	Постфиксный декремент	
	( )	Вызов функции	
	[ ]	Взятие элемента массива	
3	++	Префиксный инкремент	Справа налево
	--	Префиксный декремент	
	+	Унарный плюс	
	-	Унарный минус	
	!	Логическое НЕ	
	(*тип*)	Приведение типа	
	*	Разыменование указателя	
	&	Взятие адреса объекта	
	sizeof	Определение размера объекта в байтах	
	new, new[]	Выделение динамической памяти (C++)	

Приоритет	Оператор	Описание	Ассоциативность
	delete, delete[]	Освобождение динамической памяти (C++)	
4	.*	Указатель на член класса	Слева направо
	->*	Указатель на член класса	
5	*	Умножение	
	/	Деление	
	%	Получение остатка от целочисленного деления	
6	+	Сложение	
	-	Вычитание	
7	~	Побитовая инверсия	
	<<	Побитовый сдвиг влево	
	>>	Побитовый сдвиг вправо	
8	<	Меньше	
	<=	Меньше или равно	
	>	Больше	
	>=	Больше или равно	
9	= =	Логическое равно	
	!=	Логическое не равно	
10	&	Побитовое И (and)	
11	^	Побитовое исключающее ИЛИ (xor)	
12	|	Побитовое ИЛИ (or)	
13	&&	Логическое И	
14	||	Логическое ИЛИ	
15	?:	Тернарная условная операция	Справа налево
16	=	Присваивание	
	+=	Сложение, совмещённое с присваиванием	
	-=	Вычитание, совмещённое с присваиванием	
	*=	Умножение, совмещённое с присваиванием	

Приоритет	Оператор	Описание	Ассоциативность
	/=	Деление, совмещённое с присваиванием	
	%=	Вычисление остатка от деления, совмещённое с присваиванием	
	+<<=	Побитовый сдвиг влево, совмещённый с присваиванием	
	+>>=	Побитовый сдвиг вправо, совмещённый с присваиванием	
	+&=	Побитовое «И», совмещённое с присваиванием	
	+|=	Побитовое «ИЛИ», совмещённое с присваиванием	
	+^=	Побитовое «исключающее ИЛИ» (xor), совмещённое с присваиванием	
17	throw	Оператор создания исключения (C++)	
18	,	Оператор «запятая»	Слева направо

Таблица 5а. Математические функции из библиотеки `cmath`

Необходимо запомнить то, что операнды данных функций всегда *должны быть вещественными*, то есть *a* и *b* - числа с плавающей точкой. По умолчанию и аргументы и результаты функции имеют тип `double`.

Матем. функция	Имя функции	Название	Пример
|x|	**abs(x)**	абсолютное значение **X**	**abs(-3.0)= 3.0** **abs(5.0)= 5.0**
$\sqrt{x}$	**sqrt(x)**	квадратный корень **X**	**sqrt(9.0)=3.0**
ln x	**log(x)**	натуральный логарифм **X**	**log(1.0)=0.0**
lg x	**log10(x)**	десятичный логарифм **X**	**log10(100.)=2**
e^x	**exp(x)**	**e** в степени **X**	**exp(0.)=1**
a^x	**pow(a,x)**	**a** в степени **X**	**pow(2.,3)=8**
sin x	**sin(x)**	синус **x** (**x** задаётся в радианах)	

Матем. функция	Имя функции	Название	Пример
cos x	**cos(x)**	косинус **x (x** задаётся в радианах**)**	
tg x	**tan(x)**	тангенс **x (x** задаётся в радианах**)**	
arcsin x	**asin(x)**	Возвращает угол, синус которого равен **x**	
arccos X	**acos(x)**	Возвращает угол, косинус которого равен **x**	
arctg X	**atan(x)**	Возвращает угол, тангенс которого равен **x**	
	floor(x)	Округляет **x** до целого в меньшую сторону	**floor(12.4)=12** **floor(-2.9)=-3**
	ceil(x)	Округляет **x** до целого в большую сторону	**ceil(2.3)=3.0** **ceil(-2.3)=-2.0**
	fmod(x, y)	Возвращает остаток от деления вещественных **x / y**	**fmod(4.4, 7.5) = 4.4** **fmod(7.5, 4.4) = 3.1**

Таблица 5б. Математические функции из заголовочного файла `cstdlib`

Имя функции	Название
srand (x)	Используется для вычисления стартовой точки при генерации последовательности псевдослучайных чисел, где x типа `unsigned int`
rand ( )	Генерирует последовательность псевдослучайных чисел в диапазоне от 0 до числа RAND_MAX
RAND_MAX	— это константа, определённая в библиотеке <cstdlib>. Для Microsoft Visual Studio RAND_MAX = 32767, но оно может быть и больше, в зависимости от компилятора.

6. Ввод-вывод C++

Для управления вводом-выводом в операторах `cin` | `cout` используются:

- *флаги* форматированного ввода-вывода;
- *манипуляторы* форматирования.

Флаги позволяют установить параметры ввода-вывода, которые будут действовать во всех последующих операторах ввода-вывода до тех пор, пока не будут отменены. Манипуляторы вставляются в операторы `cin | cout` и устанавливают параметры текущего оператора ввода-вывода.

Таблица 6а. Флаги форматного вывода с примерами их использования

Флаг	Описание	Пример использования	Результат

right	Выравнивание по правой границе	`int r=-25;`
		`cout.setf(ios::right);`
		`cout.width(15);`
		`cout<<"r="<<r<<endl;`
			`r=-25`
			`Для продолжения нажмите любую клавишу . . .`

left	Выравнивание по левой границе (по умолчанию)	`double r=-25.45;`
		`cout.setf(ios::left);`
		`cout.width(20);`
		`cout<<"r="<<r<<endl;`
			`r=  -25.45`
			`Для продолжения нажмите любую клавишу . . .`

showpos	Выводить знак + для положительных чисел	`int p=29;`
		`cout.setf(ios::showpos);`
		`cout<<"p="<<p<<endl;`
			`p=+29`
			`Для продолжения нажмите любую клавишу .`

scientific	Экспоненциальная форма вывода вещественных чисел	`double p=146.673;`
		`cout.setf(ios::scientific);`
		`cout<<"p="<<p<<endl;`
			p=1.466730e+002

fixed	Фиксированная форма вывода вещественных чисел (по умолчанию)	`double p=146.673;`
		`cout.setf(ios::fixed);`
		`cout<<"p="<<p<<endl;`
			p=146.673

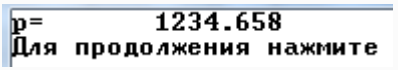
Флаги удобно использовать в тех случаях, когда следует *изменить параметры всех последующих операторов* ввода-вывода. Использование большего количества флагов для управления одним оператором ввода-вывода не совсем удобно.

Манипуляторы встраиваются непосредственно в операторы ввода-вывода. В таблице ниже приведены основные манипуляторы форматирования с примерами. Для корректного их использования необходимо подключить библиотеку iomanip (input|output manipulation) с помощью оператора #include.

Таблица 6б. Манипуляторы форматного вывода с примерами их использования

Манипулятор	Описание	Пример использования	Результат

setw(n)	Определяет ширину поля вывода в n символов	`int r=-253;`
		`cout.setf(ios::fixed);`
		`cout<<"r="<<setw(10)<<r<<endl;`
			

setprecision(n)	Определяет количество цифр (n-1) в дробной части числа	`double p=1234.6578;`
		`cout.setf(ios::fixed);`
		`cout<<"p="<<setw(15)<<`
		`setprecision(3)<<p<<endl;`
			

Манипулятор	Описание	Пример использования	Результат

left	Выравнивание по левой границе (по умолчанию)	int r=-25;
		cout.width(15);
		cout<<"r="<<setw(15)<<left<<r<<endl;
			r=-25
			Для продолжения нажмите любу

right	Выравнивание по правой границе	int r=-25;
		cout.width(15);
		cout<<"r="<<setw(15)<<right<<r<<endl;
			r= -25
			Для продолжения нажмите любую к

showpos	Выводить знак + для положительных чисел	int p=29; cout<<"p="<<showpos<<p<<endl;	p=+29
noshowpos	Не выводить знак + для положительных чисел	int p=29; cout<<"p="<<noshowpos<<p<<endl;	p=29
scientific	Экспоненциальная форма вывода вещественных чисел	double p=146.673; cout<<"p="<<scientific<<p<<endl;	p=1.466730e+002
setfill(c)	Установить символ с как заполнитель	cout<<"x="<<right<<setw(10)<<setprecision(4)<<setfill('!')<<(float) 1/7<<endl; cout<<"x="<<left<<setw(10)<<setprecision(4)<<setfill('!')<<(float) 1/7<<endl;	x=!!!!0.1429 x=0.1429!!!! Для продолжения нажм

7. Функция форматированного ввода/вывода

Компилятор C++ поддерживает работу функций ввода/вывода C printf () и scanf().

Для вывода используется функция:

int printf(const char *format, arg-list)

Функция printf() записывает в поток вывода аргументы из списка arg-list под управлением строки, на которую указывает аргумент format. *Спецификаторы формата* состоят из символа процент (%), за которым следует код формата. Команды форматирования приведены в таблице. Количество аргументов должно точно соответствовать количеству спецификаторов формата, причем следовать они должны в одинаковом порядке. Например, следующий вызов функции printf()

printf ("Hi %c %d %s", 'c', 10, "there!");

приведет к выводу «Hi c 10 there!».

Таблица7а Спецификаторы формата вывода

Код	Формат
%c	Символ типа char
%d	Десятичное число целого типа со знаком
%i	Десятичное число целого типа со знаком
%e	Научная нотация (e нижнего регистра)
%E	Научная нотация (E верхнего регистра)
%f	Десятичное число с плавающей точкой
%g	Использует код %e или %f — тот из них, который короче (при использовании %g используется e нижнего регистра)
%G	Использует код %E или %f — тот из них, который короче (при использовании %G используется E верхнего регистра)
%o	Восьмеричное целое число без знака
%s	Строка символов
%u	Десятичное число целого типа без знака
%x	Шестнадцатеричное целое число без знака (буквы нижнего регистра)
%X	Шестнадцатеричное целое число без знака (буквы верхнего регистра)
%p	Выводит на экран значение указателя
%n	Ассоциированный аргумент — это указатель на переменную целого типа, в которую помещено количество символов, записанных на данный момент
%%	Выводит символ %

Если количество аргументов меньше, чем количество команд форматирования, то вывод не определен. Если же количество аргументов больше, чем команд форматирования, то лишние ар-гументы отбрасываются.

Функция printf() возвращает количество действительно выведенных символов. Возврат отрицательной величины означает ошибку.

Команды форматирования могут содержать *модификаторы*, означающие *ширину поля, точность и флаг выравнивания влево*. Переменная целого типа, помещенная между символом процент и командой форматирования, работает как спецификатор минимальной ширины поля, заполняя поле вывода пробелами или нулями так, чтобы обеспечить указанную минимальную ширину. Если строка или число больше, чем этот минимум, они будут полностью выведены. По умолчанию заполнение производится пробелами. При выводе числовых переменных, если надо использовать заполнение нулями, помещается нуль перед спецификатором минимальной ширины поля. Например, %05d будет дополнять числа, состоящие из менее чем 5 цифр, нулями до пяти цифр.

Результат использования модификатора точности зависит от типа модифицируемой команды форматирования. Чтобы использовать модификатор точности, надо поместить десятичную точку и точность вслед за ней после количества выводимых десятичных знаков. Например, %10.4f означает вывод числа шириной минимум 10 символов с четырьмя знаками после точки. Однако при использовании совместно со спецификаторами g или G модификатор точности задает максимальное количество отображаемых значащих цифр.

Когда модификатор точности применяется к целым числам, он указывает минимальное количество отображаемых цифр. (При необходимости отображаются предшествующие нули.)

Когда модификатор точности применяется к строкам, число после десятичной точки указывает максимальную длину поля. Например, %5.7s выводит строку длиной не менее пяти и не более семи символов. Если строка длиннее, чем максимальная ширина поля, то последние символы будут урезаны.

По умолчанию вывод производится с выравниванием вправо. Это значит, что если ширина поля больше, чем выводимые данные, то данные будут размещены на правом краю поля. Можно задать режим выравнивания влево, вставив знак минус сразу после знака процент. Например, %-10.2f прижмет влево в десятизнаковом поле число с плавающей точкой с двумя знаками после запятой.

Для вывода используется функция:

int scanf(const char *format, arg-list)

Для Visual Studio 2013 и выше используется расширенная версия функции:

int scanf_s(const char *format, arg-list, int N_buf)

scanf_s принимает дополнительный аргумент, максимальный размер считываемого блока данных (N_buf). Это важно при считывании строк, чтобы не было переполнения буфера.

Функция scanf() является процедурой ввода общего назначения, считывающей данные из потока ввода. Она может считывать данные всех базовых типов и автоматически конвертировать их в нужный внутренний формат.

Таблица7б Спецификаторы формата ввода

Код	Значение
%c	Считать один символ
%d	Считать десятичное число целого типа
%i	Считать десятичное число целого типа
%e	Считать число с плавающей запятой
%f	Считать число с плавающей запятой
%g	Считать число с плавающей запятой
%o	Считать восьмеричное число
%s	Считать строку
%x	Считать шестнадцатеричное число
%p	Считать указатель
%n	Принимает целое значение, равное количеству считанных до текущего момента символов
%u	Считывает беззнаковое целое
%[]	Просматривает набор символов
%%	Считывает символ %

Например, %s считывает строку, а %d считывает переменную целого типа.

Строка формата считывается слева направо, при этом устанавливается соответствие между ко-дами формата и аргументами из списка аргументов.

Специальные символы в управляющей строке заставляют scanf() пропускать один или больше специальных символов во входном потоке. Специальные символы — это пробел, табуляция или новая строка. Один специальный символ в управляющей строке заставляет scanf() считывать, не запоминая, любое количество (включая нуль) идущих подряд специальных

символов из входного потока, пока не встретится символ, не являющийся специальным символом.

Наличие обычного символа заставляет scanf() считать и отбросить соответствующий символ. Например, "%d,%d" заставляет scanf() считать целое число, считать и отбросить запятую и затем считать еще одно целое число. Если указанный символ не обнаружен во входном потоке, scanf() останавливается.

Все переменные, используемые для приема значений с помощью функции scanf(), должны отыс-киваться по их адресам. Это значит, что все аргументы функции должны быть указателями на переменные. Таким образом, С создает возможность передачи по ссылке, и это позволяет функ-ции изменять содержимое аргумента.

Например, чтобы считать целое число и присвоить его значение переменной count, необходи-мо воспользоваться следующим обращением к scanf():

```
scanf("%d", &count);
```

Хотя пробелы, символы табуляции и новых строк используются как разделители полей, они считываются как любой другой символ при вводе одиночного символа. Например, при входном потоке x y функция

```
scanf("%c%c%c", &a, &b, &c);
```

поместит символ x в переменную a, пробел — в переменную b и y — в переменную c.

Надо быть внимательным: любые другие символы в управляющей строке — включая пробелы, символы табуляции и новых строк — используются для указания и отбрасывания символов из входного потока. Например, при входном потоке 10t20 функция

```
scanf ("%st%s", &x, &y);
```

поместит 10 в x, а 20 в y. Символ t будет отброшен, поскольку в управляющей строке имеется t.

8. Описание массивов в программе на C++

При использовании простых переменных каждой области памяти для хранения данных соответствует свое имя. Если с группой величин одинакового типа требуется выполнять однообразные действия, им дают одно имя, а различают по порядковому номеру. Это позволяет компактно записывать множество операций с помощью циклов. *Конечная именованная последовательность однотипных величин называется массивом*. Различают два типа массивов по способу выделения памяти – *статические и динамические.*

Более простыми являются ***статические массивы***. Описание статического массива в программе отличается от описания простой переменной наличием после имени квадратных скобок, в которых задается количество элементов массива (размерность):

```
float a[10]; // описание массива из 10 вещественных чисел
```

Элементы массива нумеруются с нуля. Инициализирующие значения для массивов записываются в фигурных скобках. Значения элементам присваиваются по порядку. Если элементов в массиве больше, чем инициализаторов, элементы, для которых значения не указаны, обнуляются:

```
int b[5] = {3. 2. 1}; / / b[0]=3. b[l]=2. b[2]=l. b[3]=0.  b[4]=0
```

Размерность массива вместе с типом его элементов определяет объем памяти, необходимый для размещения массива, которое выполняется на этапе компиляции, поэтому *размерность может быть задана только целой положительной константой или константным выражением*. Если при описании массива не указана размерность, должен присутствовать инициализатор, в этом случае компилятор выделит память по количеству инициализирующих значений.

Для доступа к элементу массива после его имени *указывается номер элемента (индекс)* в квадратных скобках. В следующем примере подсчитывается сумма элементов массива.

```
#include <iostream>
int main(){
const int n = 10;
int i, sum;
int b[n] = {3. 4. 5. 4. 4};
for (i = 0, sum = 0; i<n; i++) sum += b[i];
cout << "Сумма элементов; " << sum;
return 0;
}
```

Размерность статических массивов предпочтительнее задавать с помощью именованных констант, как это сделано в примере, поскольку при таком подходе для ее изменения достаточно скорректировать значение константы всего лишь в одном месте программы.

ВНИМАНИЕ!

- Последний элемент массива имеет номер, на единицу меньший заданной при его описании размерности.
- При обращении к элементам массива *автоматический контроль выхода индекса за границу массива не производится*, что может привести к ошибкам.

Размер статического массива может быть вычислен в программе с помощью оператора `sizeof(аргумент)`, который сообщает размер аргумента в байтах (Табл. 3f Приложения). Пример применения оператора `sizeof()` приведён в *Примере 7* Л/Р №1 и *Примере 4* Л/Р №3.

Идентификатор массива является константным указателем на его нулевой элемент. Например, для массива приведённой выше программы имя `marks` — это то же самое, что `& b`[0], а к i-му элементу массива можно обратиться, используя выражение `*(b+1)`. Можно описать указатель, присвоить ему адрес начала массива и *работать с массивом через указатель*. Следующий фрагмент программы копирует все элементы массива *a* в массив *b*:

```
int a[100]. b[100];
int *pa = a; // или int *p = &a[0];
int *pb = b;
for (int i = 0: i < 100: i++)
*pb++ =>pa++; // или pb[i] = pa[i];
```

Динамические массивы создают с помощью операции new, при этом необходимо указать тип и размерность, например:

```
int n = 100;
float *p = new float [n];
```

В этой строке создается переменная-указатель на `float`, в динамической памяти отводится непрерывная область, достаточная для размещения 100 элементов вещественного типа, и адрес ее начала записывается в указатель p. *Динамические массивы нельзя при создании инициализировать*, и они не обнуляются.

Преимущество динамических массивов состоит в том, что размерность может быть переменной, то есть объем памяти, выделяемой под массив, определяется на этапе выполнения программы. Доступ к элементам динамического массива осуществляется точно так же, как к статическим, например, к элементу номер 5 приведенного выше массива можно обратиться как `p`[5] или *(`p+5`).

Память, зарезервированная под динамический массив с помощью `new[]`, должна освобождаться оператором `delete []`, например:

```
delete [ ] p;
```

При несоответствии способов выделения и освобождения памяти результат не определен. Размерность массива в операции `delete` не указывается, но квадратные скобки обязательны.