Министерство науки и высшего образования Российской Федерации

Калужский филиал
федерального государственного бюджетного образовательного
учреждения высшего образования
**«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»**
(КФ МГТУ им. Н.Э. Баумана)

Ю.С. Белов, С.С. Гришунов

МОДЕЛИРОВАНИЕ ОПЕРАЦИЙ НАД ДЛИННЫМИ ЧИСЛАМИ

Методические указания к выполнению домашней работы
по курсу «Типы и структуры данных»

Калуга – 2019

УДК 004.62
ББК 32.972.5
Б435

Методические указания составлены в соответствии с учебным планом КФ МГТУ им. Н.Э. Баумана по направлению подготовки 09.03.04 «Программная инженерия» кафедры «Программного обеспечения ЭВМ, информационных технологий».

Методические указания рассмотрены и одобрены:

- Кафедрой «Программного обеспечения ЭВМ, информационных технологий» (ИУ4-КФ) протокол № 51.4/5 от «23» января 2019 г.

Зав. кафедрой ИУ4-КФ _____ к.т.н., доцент Ю.Е. Гагарин

- Методической комиссией факультета ИУ-КФ протокол №7 от «18» 04 2019 г.

Председатель методической комиссии факультета ИУ-КФ _____ к.т.н., доцент М.Ю. Адкин

- Методической комиссией КФ МГТУ им.Н.Э. Баумана протокол № 4 от «5» 02 2019 г.

Председатель методической комиссии КФ МГТУ им.Н.Э. Баумана _____ д.э.н., профессор О.Л. Перерва

Рецензент:
к.т.н., доцент кафедры ИУ6-КФ _____ А.Б. Лачихина

Авторы
к.ф.-м.н., доцент кафедры ИУ4-КФ _____ Ю.С. Белов
ассистент кафедры ИУ4-КФ _____ С.С. Гришунов

Аннотация

Методические указания к выполнению домашнего задания по курсу «Типы и структуры данных» содержат общие сведения о целых и вещественных числах, длинной арифметики и алгоритмах выполнения арифметических операций над длинными числами.

Предназначены для студентов 2-го курса бакалавриата КФ МГТУ им. Н.Э. Баумана, обучающихся по направлению подготовки 09.03.04 «Программная инженерия».

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Настоящие методические указания составлены в соответствии с программой проведения домашних заданий по курсу «Типы и структуры данных» на кафедре «Программное обеспечение ЭВМ, информационные технологии» факультета «Информатика и управление» Калужского филиала МГТУ им. Н.Э. Баумана.

Методические указания к выполнению домашней работы по курсу «Типы и структуры данных» содержат краткое описание целых и вещественных чисел, алгоритмы арифметических операций над длинными числами.

Предназначены для студентов 2-го курса бакалавриата КФ МГТУ им Н.Э. Баумана, обучающихся по направлению подготовки 09.03.04 «Программная инженерия».

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ ВЫПОЛНЕНИЯ

Целью выполнения домашней работы является формирование практических навыков моделирования операций над длинными числами.

Основными задачами выполнения домашней работы являются:

1. Познакомиться с представлением чисел в памяти компьютера.
2. Создать собственную модель для представления длинного числа в памяти компьютера.
3. Научиться составлять и реализовывать алгоритмы для арифметических операций над длинными числами.
4. Смоделировать математическую операцию с длинными числами согласно варианту.

Результатами работы являются:

- Программа, реализующая индивидуальное задание
- Подготовленный отчет

КРАТКАЯ ХАРАКТЕРИСТИКА ОБЪЕКТА ИЗУЧЕНИЯ, ИССЛЕДОВАНИЯ

В данном домашней работе рассматриваются два простых типа данных: *целые* и *вещественные* числа.

Целые числа

Целые числа являются простейшими числовыми данными, с которыми оперирует ЭВМ. Для целых чисел существуют два представления: беззнаковое (только для неотрицательных целых чисел) и со знаком. Очевидно, что отрицательные числа можно представлять только в знаковом виде. Целые числа в компьютере хранятся в формате с фиксированной запятой.

В общем случае под целое число можно отвести любое число соседних байт памяти, однако система команд компьютера поддерживает работу с числами только размеров в байт, слово и двойное слово (если числа занимают иное количество байт, то все операции над ними надо реализовывать самому программисту).

В компьютере делается различие между целыми со знаком и целыми без знака (неотрицательных). Это объясняется тем, что в ячейках одного и того же размера можно представить больший диапазон беззнаковых чисел, чем неотрицательных знаковых. Например, в байте можно представить беззнаковые числа от *0* до *255*, а неотрицательные знаковые числа – только от *0* до *127*. Поэтому, если известно, что некоторая числовая величина является неотрицательной, то выгоднее рассматривать ее как беззнаковую, чем как знаковую.

Целые числа со знаком также представляются в виде байта, слова и двойного слова. Как байт можно представить числа от *-128* до *+127*, как слово – от *-32768* до *+32767*, как двойное слово – от *-2147483648* до *+2147483647*. Знак содержится в старшем бите числа. Ноль в этом бите соответствует положительному числу, а единица – отрицательному. В компьютере знаковые числа записываются в дополнительном коде: неотрицательное число записывается так же, как и беззнаковое число, а отрицательное число x представляется беззнаковым числом $2^k - |x|$, где k – количество разрядов в ячейке, отведенной под число.

Вещественные числа

Вещественные числа обычно представляются в виде чисел с плавающей запятой. Числа с плавающей запятой — один из возможных способов представления действительных чисел, который является компромиссом между точностью и диапазоном принимаемых значений, его можно считать аналогом экспоненциальной записи чисел, но только в памяти компьютера.

Число с плавающей запятой состоит из набора отдельных двоичных разрядов, условно разделенных на так называемые знак (англ. sign), порядок (англ. exponent) и мантиссу (англ. mantis). В наиболее распространенном формате (стандарт IEEE 754) число с плавающей запятой представляется в виде набора битов, часть из которых кодирует собой мантиссу числа, другая часть — показатель степени, и еще один бит используется для указания знака числа (0 — если число положительное, 1 — если число отрицательное). При этом порядок записывается как целое число в коде со сдвигом, а мантисса — в нормализованном виде, своей дробной частью в двоичной системе счисления.

Знак — один бит, указывающий знак всего числа с плавающей точкой. Порядок и мантисса — целые числа, которые вместе со знаком дают представление числа с плавающей запятой

Вещественные числа обычно хранятся и используются в формате с плавающей точкой в следующем виде: $x = M*E^p$, где M – мантисса со знаком, E – основание (10 или 16), p – целый порядок со знаком. Вместо порядка часто используется характеристика, получающаяся прибавлением к порядку такого смещения, чтобы характеристика была всегда неотрицательна.

Порядок также иногда называют экспонентой или просто показателем степени.

При этом лишь некоторые из вещественных чисел могут быть представлены в памяти компьютера точным значением, в то время как остальные числа представляются приближенными значениями.

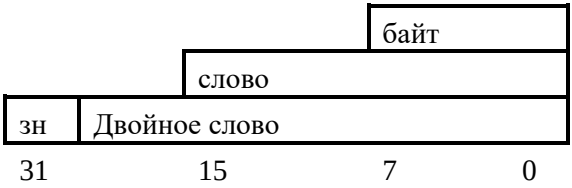
Более простым вариантом представления вещественных чисел является вариант с фиксированной точкой, когда целая и вещественная части хранятся отдельно. Например, на целую часть отводится всегда

X бит и на дробную отводится всегда Y бит. Такой способ в архитектурах процессоров не присутствует. Отдается предпочтение числам с плавающей запятой, как компромиссу между диапазоном допустимых значений и точностью.

Форматы целых и вещественных чисел представлены на рисунке 1.

Целые без знака

Целые со знаком



Вещественное число

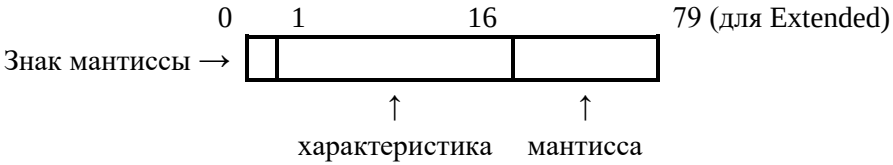


Рис. 1.

Диапазон значений чисел с плавающей запятой

Диапазон чисел, которые можно записать данным способом, зависит от количества бит, отведенных для представления мантиссы и показателя. Пара значений показателя (когда все разряды нули и когда все разряды единицы) зарезервирована для обеспечения возможности представления специальных чисел. К ним относятся ноль, значения NaN (Not a Number, "не число", получается как результат операций типа деления нуля на ноль) и $\pm\infty$.

Таблица 1 только лишь примерно указывает границы допустимых значений, без учета возрастающей погрешности с ростом абсолютного значения и существования денормализованных чисел.

Таблица 1

Название в IEEE 754	Название типа переменной в Си	Диапазон значений	Бит в мантиссе	Бит на переменную
Half precision	-	$6{,}10\times10^{-5}..65504$	11	16
Single presicion	float	$3{,}4\times10^{-38}..3{,}4\times10^{38}$	23	32
Double precision	double	$1{,}7\times10^{-308}..1{,}7\times10^{308}$	53	64
Extended precision	На некоторых архитектурах (например в сопроцессоре Intel) long double	$3{,}4\times10^{-4932}..3{,}4\times10^{4932}$	65	80

Длинная арифметика

В языках программирования всегда существует несколько стандартных типов переменных, которые представляют целое число. Зачем нужны разные целые числа, и чем они отличаются?

Таблица 2

Тип в C++	Диапазон значений	Размер одной переменой этого типа
char	-128..127	1 байт
unsigned char	0..255	1 байт
Short	-32768..32767	2 байта
unsigned short	0..65535	2 байта
int	-2147483648..2147483647	4 байта
unsigned int	0..4294967295	4 байта
long long	$-2^{63}..2^{63}-1$	8 байт
unsigned long long	$0..2^{64}-1$	8 байт

Как видно из таблицы 2 разные типы имеют разный размер, и поэтому могут представлять целые числа из разных диапазонов. Чем больше размер переменной, тем шире диапазон чисел, который может быть представлен этой переменной.

Если вы работаете с переменной типа *integer* (*int*) и число, которое хранится в переменной становится больше 2147483647 или меньше -2147483648, то произойдет переполнение. В C++ и на Pascal (при выключенном ключе компиляции {$Q-}) программа продолжит свое выполнение, но число в переменной будет записано неверное. Если в начале программы на Pascal написать {$Q+}, то при возникновении такой ошибки программа аварийно прекратит свое выполнение. Эта возможность помогает искать ошибки, связанные с переполнением. В C++ такая возможность не предусмотрена.

Несмотря на то, что с помощью стандартных типов можно представить достаточно большие числа, иногда возникает необходимость работать с числами намного больше.

Известно, что арифметические действия, выполняемые компьютером в ограниченном числе разрядов, не всегда позволяют получить точный результат. Более того, мы ограничены размером (величиной) чисел, с которыми можем работать. А если нам необходимо выполнить арифметические действия над очень большими числами, например 30!=265252859812191058636308480000000? В таких случаях мы сами

должны позаботиться о представлении чисел и о точном выполнении арифметических операций над ними.

Числа, для представления которых в стандартных компьютерных типах данных не хватает количества двоичных разрядов, называются иногда «длинными». В этом случае программисту приходится самостоятельно создавать подпрограммы выполнения арифметических операций.

Стандартных типов для хранений чисел, состоящих из 20 и более цифр, в рассматриваемых нами языках программирования нет, поэтому первое, что нужно обсудить — это каким образом мы будем хранить наше число в памяти. Самый естественный и удобный способ — это хранить цифры числа в массиве. Одна цифра — это число от 0 до 9, а значит, ее можно хранить в переменной любого размера. При таком способе хранения мы можем работать с числами произвольного размера. В ячейке номер 0, будем хранить количество цифр в данном числе.

```
#define DMAX 100
typedef int thuge[DMAX];
```

Сложение длинных чисел

Каждый из нас уже сталкивался с подобной ситуацией еще в первом классе. Выучив таблицу сложения, мы совершенно не умели складывать двузначные числа. Для решения этой проблемы нас научили складывать числа «столбиком». Сначала складываются младшие разряды. Если сумма оказывается больше 9, то мы запоминаем «один в уме», то есть, прибавляем единицу в старший разряд. В результат записывается последняя цифра суммы:

```
 1234
  298
------
 1532
```

При сложении «столбиком» двух чисел с разным количеством цифр, числа записываются так, чтобы последние цифры стояли друг под другом. Если мы храним число в массиве, то нам достаточно неудобно

передвигать число с меньшим количеством знаков таким образом, чтобы младшие разряды стояли на одинаковых позициях. Кроме всего прочего результат сложения может состоять из большего количества цифр, чем каждое из слагаемых. В связи с этим нам может потребоваться записать цифру в самое начало (т. е. перед первой). Так как первая цифра записана в ячейке номер 1, то придется сдвинуть все число на одну позицию вправо и только после этого записать старшую цифру. В таком случае реализация становится очень запутанной с большим количеством случаев и частыми сдвигами числа по массиву. Все это может привести к замедлению программы и большому количеству ошибок при реализации. Чтобы избежать этих трудностей, лучше хранить число «перевернутым», то есть, в первой ячейке массива будет записана младшая цифра (единицы), во второй будут записаны десятки и так далее. Это решает сразу все перечисленные проблемы. Во-первых, все числа теперь записаны так, что их последние цифры находятся друг под другом, во-вторых, чтобы добавить к числу цифру слева, нужно просто дописать ее в конец массива.

Существуют разные системы счисления. В принципе мы можем работать в любой из них. Но нужно помнить, что перенос происходит, если сумма стала больше или равна основания системы счисления. Чтобы работать в системе счисления *base*, нужно заменить в программе все *10* на *base*. (В таком случае желательно завести константу *base*.) Для достижения максимальной скорости программы нужно выбирать основание системы исчисления равное некоторой степени двойки. При таком выборе можно реализовать более быстрые операции, связанные с *base*. Но в таком случае возникают некоторые трудности при вводе и выводе числа. Для ввода и вывода придется переводить число из одной системы счисления в другую. Это достаточно трудоемкая операция, поэтому для нас будут представлять интерес системы счисления с основанием 10^K (для некоторого натурального числа K). Такая система счисления, фактически, означает, что в одной ячейке массива будет храниться не одна, а сразу K цифр. Очевидно, что операция сложения будет работать в K раз быстрее, так как в числах будет в K раз меньше цифр.

Рассмотрим арифметическую операцию сложения, применяемую в длинной арифметике. Алгоритм этого нехитрого арифметического действия, на удивление, простой. Он выглядит так:

```
void add(thuge a, thuge &b, thuge &c){
	//функция прибавляет к числу a число b
	if (a[0] < b[0]) a[0] = b[0];
	//складывать нужно до размера большего числа
	int r = 0;
	/*r - обозначает сколько у нас "в уме"
	 при сложение младших цифр в уме у нас 0*/
	for(int i = 1; i <= a[0]; ++i) {
		a[i] += b[i] + r;
		//сумма очередных цифр и переноса
		if (a[i] >= 10) {
		//случай, когда происходит перенос
			r = 1;
			a[i] -= 10;
		} else {
		//случай, когда переноса не происходит
			r = 0;
		}
	}
	//если после сложения остался перенос,
	//то нужно добавить еще одну цифру
	if (r > 0) {
		a[0]++;
		a[a[0]] = r;
	}
	c = a;
}
```

Вычитание длинных чисел

Вычитание в столбик практически аналогично. Если при вычитании двух цифр результат получается отрицательным, то мы занимаем единицу из старшего разряда. Мы будем работать только с неотрицательными числами, поэтому при вычитании будем подразумевать, что разность неотрицательна. В отличие от сложения количество цифр в разности может уменьшиться, поэтому нужно будет пересчитать количество цифр.

```
void subtract(thuge &a, thuge &b) {
	//функция вычитает из числа a число b
	//r - обозначает был ли заем из текущего разряда
	int r = 0;
	//заем из младшего разряда отсутствует
	for(int i = 1; i <= a[0]; ++i) {
		a[i] -= b[i] + r;
		//разность очередных цифр с учетом заема
		if (a[i] < 0) {
		//случай, когда происходит заем
		//из следующего разряда
			r = 1;
			a[i] += base;
		} else {
		//случай, когда заем не происходит
			r = 0;
		}
	}
	/*Разность может содержать меньше цифр, поэтому нужно
	при необходимости уменьшить количество цифр*/
	while (a[0] > 1 && a[a[0]] == 0) {
		--a[0];
	}
}
```

Сложение и вычитание работает за длину наибольшего числа. Числа, которые встречаются в процессе вычислений, не превосходят по модулю *2 * base*. Значит в одной ячейке массива при использовании типа longint (int) можно хранить до 9 десятичных цифр (то есть основание системы исчисления может быть до 10^9).

Сравнение чисел

Первое на что мы обратим внимание, когда будем сравнивать числа, будет количество цифр в них. Если количество цифр различно, то больше то из них, которое содержит больше цифр. Если количество цифр одинаково, то нужно сравнивать, начиная со старшей цифры. При обнаружении первого различия (т. е. самая старшая цифра, в которой есть различие), можно определить какое из чисел больше. Если два числа не имеют различий, то они равны.

```
int compare(thuge &a, thuge &b) {
/*
a < b => -1
a > b =>  1
a = b =>  0
*/
//сравнение по количеству цифр
    if (a[0] < b[0]) {
          return -1;
    }
    if (a[0] > b[0]) {
          return 1;
    }
    /*сравнение в случае одинакового количества цифр*/
    for(int i = a[0]; i >= 1; --i) {
          if (a[i] < b[i]) {
                return -1;
          }
          if (a[i] > b[i]) {
                return 1;
          }
    }
    return 0;
}
```

Ввод и вывод длинного числа

Так как пользовательский тип, определяющий длинное число, не является стандартным, то его нельзя прочитать, используя просто функцию *read* и выводить с помощью функции *write*. Для того, чтобы вывести число, записанное в десятичной системе исчисления достаточно последовательно вывести цифры последовательно, начиная со старшей. Нужно быть внимательным при выводе цифр, состоящих из маленького количества цифр, так как к таким цифрам при выводе необходимо дописать лидирующие нули. Обратите внимание, что к самой первой цифре приписывать лидирующие нули не нужно.

```
const char * digitFormat = "%.d";
/*формат вывода числа ровно из 4 цифр,
недостающие цифры дополняются нулями*/
```

```
void writeHuge(thuge &a) {
	printf("%d", a[a[0]]);
	for(int i = a[0] - 1; i >= 1; --i) {
		printf(digitFormat, a[i]);
	}
	printf("\n");
}
```

При чтении длинного числа нужно помнить, что цифры записываются в обратном порядке. Но в случае системы исчисления большей 10, «десятичные цифры» внутри цифры большей системы исчисления идут в прямом порядке. Например, число 12345678 в 100-ричной системе исчисления будет записано как: 78, 56, 34, 122.

```
const int digit = 1;
/*количество цифр в одной ячейке массива*/
void readHuge(thuge &a) {
	char s[DMAX * digit + 1];
	scanf("%s", s);//чтение строки
	memset(a, 0, sizeof(a));
	int len = strlen(s);
	a[0] = (len - 1) / digit + 1;
	/*вычисление количества цифр
	в нужной системе исчисления*/
	for(int i = 1; i <= a[0]; ++i) {
		/*вычисление iтой цифры*/
		for(int j = digit; j >= 1; --j) {
			int pos = len - (i - 1) * digit - j;
			/*вычисление позиции, из которой нужно
			дописать к iтой цифре. Помните, что
			массив s индексирован с 0*/
			if (pos >= 0) {
				a[i] = a[i] * 10 + (s[pos]-'0');
			}
		}
	}
}
```

Умножение длинного числа на короткое

Говоря об умножении чисел, стоит рассмотреть два случая: умножение длинного числа на короткое (т. е. число, которое помещается в стандартный тип) и умножение двух длинных чисел.

Умножение на короткое число будет схоже с умножением на однозначное число. Будем последовательно умножать короткое число на каждую цифру длинного числа, начиная с младшей цифры, записывать результат и некоторую часть произведения переносить в следующий разряд.

```
void multiply(thuge &a, int b) {
	//функция умножает число a на короткое число b
	//r - обозначает перенос в текущий разряд
	int r = 0;
	//перенос в младший разряд отсутствует
	for(int i = 1; i <= a[0]; ++i) {
		a[i] = a[i] * b + r;
		//произведение очередной цифры и короткого
		//числа с учетом переноса в текущий разряд
		r = a[i] / base;
		//вычисление переноса в следующий разряд
		a[i] -= r * base;
	}
	/*Если после умножения остался еще перенос, то
	нужно добавить еще цифру. Может потребоваться
	добавить несколько цифр, если число b больше base*/
	while (r > 0) {
		a[0]++;
		a[a[0]] = r % base;
		r = r / base;
	}
}
```

Деление длинного числа на короткое

В школе, все мы так же изучали деление столбиком. Мы будем рассматривать деление на короткое число b. При делении столбиком сначала выписывается старшая цифра, эту цифру делят на b. Частное дописывают к результату, а остаток пишут ниже. После этого к остатку приписывают следующую цифру, полученное значение делят на b. Аналогично, частное дописывают к ответу, а остаток пишут ниже. Процесс продолжается пока все цифры не будут использованы.

```
4531  | 7
42     ------
----  | 647
 33
 28
 ----
  51
  49
  ---
   2
```

Остаток, к которому уже нельзя приписать цифру, будет остатком от деления.

При умножении двух длинных чисел в столбик, первое число последовательно умножается на каждую цифру. Результат умножения на *i*-тую цифру прибавляется к общему результату со сдвигом на $i - 1$.

Деление длинного числа на короткое и умножение длинного числа на короткое работает за линейное время относительно длины числа. Умножение длинных чисел работает за время пропорциональное произведению их длин.

```
int divide(thuge a, int b, thuge &c) {
      /*функция делит число a на короткое число b и
      возвращает остаток от деления*/
      int r = 0; //изначально остаток 0
      for(int i = a[0]; i >= 1; --i) {
      /*цикл от старших цифр к младшим*/
            r = r * base + a[i];
            //приписывание очередной цифры
            a[i] = r / b;
            //запись частного в результат
            r %= b;
            //пересчет остатка
      }
      /*Частное может содержать меньше цифр,
      поэтому нужно при необходимости уменьшить
      количество цифр*/
      while (a[0] > 1 && a[a[0]] == 0) {
            --a[0];
      }
      c = a;
      return r;
}
```

Умножение длинных чисел

При умножении двух длинных чисел в столбик, первое число последовательно умножается на каждую цифру. Результат умножения на i-тую цифру прибавляется к общему результату со сдвигом на i – 1.

```
void multiply(thuge &a, thuge &b, thuge &c) {
//функция умножает число a на число b
      memset(c, 0, sizeof(c)); //заполнение массива 0
      /*c - результат умножения. В данном случае нельзя
      записывать результат в тот же массив.*/
      for(int i = 1; i <= a[0]; ++i) {
        int r = 0, j;
        for(j = 1; j <= b[0] || r > 0; ++j) {
        //пока есть перенос или в b есть еще цифры
            c[i + j - 1] += a[i] * b[j] + r;
            /*при умножении на предыдущие цифры в c уже
            записано некоторое значение, поэтому нужно
            прибавлять, а не присваивать*/
            r = c[i + j - 1] / base;
            c[i + j - 1] -= r * base;
        }
      }
      c[0] = a[0] + b[0];
      //максимально возможное количество цифр в ответе
      //но цифр может оказаться меньше
      while (c[0] > 1 && c[c[0]] == 0) {
            --c[0];
      }
}
```

Выбирая основание системы исчисления, нужно добиваться максимальной скорости работы, т. е. брать как можно большую систему исчисления, с одной стороны и не забывать о том, что стандартные типы представляют ограниченное множество чисел. Например, если в вашей программе используется умножение длинных чисел то, чтобы избежать переполнения число должно (base – 1) * base должно помещаться в базовый тип. При делении или умножении на короткое число b, число b * base + base должно помещаться в базовый тип. Не трудно заметить, что деление длинного числа на короткое и умножение длинного числа на короткое работает за линейное время

относительно длины числа. Умножение длинных чисел работает за время пропорциональное произведению их длин.

Квадратный корень

Вычисление квадратного корня — это достаточно сложная операция. Здесь под квадратным корнем числа y мы будем подразумевать наибольшее число x, такое, что $x^2<=y$. Мы делаем так, потому что работаем только с целыми числами. Функция x^2 монотонно возрастает, т. е. чем больше x, тем больше x^2. Поэтому для нахождения квадратного корня можно применить бинарный поиск. Рассмотрим середину m отрезка $[l, r]$, в котором мы производим поиск (изначально можно выбрать отрезок $[0, y]$). Если $m^2>y$, то дальнейший поиск нужно проводить в отрезке $[l, m - 1]$, иначе в отрезке $[m, r]$. Для реализации вычисление квадратного корня нам потребуется: сложение длинных чисел, деление длинного числа на короткое (эти операции нужны для нахождения середины отрезка), а также произведение длинных чисел (для вычисления m^2) и сравнение длинных чисел. Время работы этого алгоритма можно оценить, как куб длины числа y. Если длина числа y равна l, то выполниться порядка l шагов бинарного поиска, в каждом из которых будет выполнено произведение двух длинных чисел длины порядка l.

Алгоритм Евклида

Нахождение наибольшего общего делителя с помощью классического алгоритма довольно трудно для реализации с длинными числами. При реализации классического алгоритма нам потребуется производить деление двух длинных чисел, что мы вообще не рассматривали в связи со сложностью этого алгоритма. Мы же рассмотрим способ найти НОД, пользуясь лишь операциями, которые мы рассмотрели. Для этого рассмотрим 3 случая:

Оба числа четны ($2n, 2k$)

Оба числа нечетны ($2n + 1, 2k + 1$)

Числа разной четности ($2n, 2k + 1$)

В первом случае воспользуемся соображением $НОД(2n, 2k) = 2 * НОД(n, k)$. Во втором случае $НОД(2n + 1, 2k + 1) = НОД(2n – 2k, 2k +$

1). В третьем случае *НОД(2n, 2k + 1) = НОД(n, 2k + 1)*. Как мы видим при таком способе нахождения *НОД* нам потребуются только операции: деление на короткое (точнее на 2), вычитание, умножение на короткое (точнее на 2). Пункты 1 и 3 уменьшают хотя бы одно из чисел в 2 раза, а это может произойти не больше чем log этого числа. Log числа пропорционален его длине. Пункт 2 сразу же приводит нас в ситуацию 3, поэтому можно говорить, что каждые 2 шага алгоритма одно из чисел уменьшиться вдвое. На каждом шаге выполняется операция, требующая линейное время. Следовательно, время работы алгоритма пропорционально квадрату длины чисел.

ПРИМЕНЕНИЕ ДЛИННОЙ АРИФМЕТИКИ

Задача №1. Смоделируем операцию умножения двух целых чисел длиной до 30 десятичных цифр каждое

Итак, видим, что ни один из стандартных типов данных нам не подходит, следовательно, нам нужно реализовать свой тип данных для работы с такими числами. В таком случае для нас самый удобный способ — это хранить цифры числа в массиве. Одна цифра — это число от 0 до 9, а значит, ее можно хранить в переменной любого размера. При таком способе хранения мы можем работать с числами произвольного размера. В ячейке номер 0, будем хранить количество цифр в данном числе.

Зададим числа:

```
#define DMAX 30
typedef int thuge[DMAX];

thuge a, b, c;
```

Теперь нам нужно определить способ ввода/вывода таких чисел. Так как пользовательский тип, определяющий длинное число, не является стандартным, то его нельзя прочитать, используя просто функцию *read* и выводить с помощью функции *write*. Для того, чтобы вывести число, записанное в десятичной системе исчисления достаточно последовательно вывести цифры последовательно, начиная со старшей.

```
const char * digitFormat = "%.d";
/*формат вывода числа ровно из 4 цифр,
недостающие цифры дополняются нулями*/

void writeHuge(thuge &a) {
      printf("%d", a[a[0]]);
      for(int i = a[0] - 1; i >= 1; --i) {
            printf(digitFormat, a[i]);
      }
      printf("\n");
  }
```

Однако условие задачи накладывает небольшие ограничения на вывод чисел, поэтому нам придется модернизировать наш алгоритм.

```
const char * digitFormat = "%.d";
const int m = 30;
void writeHuge(thuge &a) {
      printf("0.");
      printf("%d", a[a[0]]);
      int N = 1;
      for (int i = a[0] - 1; i >= 1; --i) {
            printf(digitFormat, a[i]);
            N++;
            if (N >= m)
                  break;
      }
      printf("*10^");
      printf(digitFormat, N);
      printf("\n");
  }
```

Здесь m – это длина мантисы, а N – величина порядка.

При чтении длинного числа нужно помнить, что цифры записываются в обратном порядке.

```
const int digit = 1;
/*количество цифр в одной ячейке массива*/
void readHuge(thuge &a) {
      char s[DMAX * digit + 1];
      scanf("%s", s);//чтение строки
      memset(a, 0, sizeof(a));
      int len = strlen(s);
      a[0] = (len - 1) / digit + 1;
      /*вычисление количества цифр в нужной системе
      исчисления*/
      for(int i = 1; i <= a[0]; ++i) {
            /*вычисление iтой цифры*/
            for(int j = digit; j >= 1; --j) {
                  int pos = len - (i - 1) * digit - j;
                  /*вычисление позиции, из которой нужно
                  дописать к iтой цифре. Помните, что
                  массив s индексирован с 0*/
```

```
                    if (pos >= 0) {
                          a[i] = a[i] * 10 + (s[pos] - '0');}
                    }
      }
   }
```

После того как мы определились со способом хранения, способами ввода и вывод, нам надо реализовать умножение длинных чисел. При умножении двух длинных чисел в столбик, первое число последовательно умножается на каждую цифру. Результат умножения на i-тую цифру прибавляется к общему результату со сдвигом на i – 1.

```
const int base = 10;

void multiply(thuge &a, thuge &b, thuge &c) {
//функция умножает число a на число b
      memset(c, 0, sizeof(c)); //заполнение массива 0
      /*c - результат умножения. В данном случае нельзя
записывать результат в тот же массив.*/
      for(int i = 1; i <= a[0]; ++i) {
        int r = 0, j;
        for(j = 1; j <= b[0] || r > 0; ++j) {
        //пока есть перенос или в b есть еще цифры
            c[i + j - 1] += a[i] * b[j] + r;
            /*при умножении на предыдущие цифры в c уже
            записано некоторое значение, поэтому нужно
            прибавлять, а не присваивать*/
            r = c[i + j - 1] / base;
            c[i + j - 1] -= r * base;
        }
      }
      c[0] = a[0] + b[0];
      //максимально возможное количество цифр в ответе
      //но цифр может оказаться меньше
      while (c[0] > 1 && c[c[0]] == 0) {
            --c[0];
      }
   }
```

Итак, приступим к реализации. Введем числа, используя процедуру чтения больших чисел:

```
readHuge(a);
readHuge(b);
```

Затем перемножим эти числа:

```
multiply(a, b, c);
```

Для выдачи результата в форме (zn)0.m E N (где длина мантиссы m – до 30 значащих цифр, а величина порядка N – до 5 цифр) используем наш модифицированный метод:

```
writeHuge(c);
```

Задача решена.

Задача №2 Смоделировать операцию вычисления среднего арифметического двух целых чисел длиной до 30 десятичных цифр. Результат выдать в форме (zn)0.m1 E N1.

Итак, формат хранения длинных чисел, операции ввода/вывода у нас определены еще в задаче №1. Следовательно, для выполнения поставленной задачи нам необходимо разобрать операции сложения длинных чисел и деления длинного на короткое.

Начнем с первого. Итак, рассмотрим алгоритм.

```
void add(thuge &a, thuge &b){
//функция прибавляет к числу a число b
      if (a[0] < b[0]) a[0] = b[0];
      //складывать нужно до размера большего числа
      int r = 0;
      /*r - обозначает сколько у нас "в уме"
       при сложение младших цифр в уме у нас 0*/
      for(int i = 1; i <= a[0]; ++i) {
            a[i] += b[i] + r;
            //сумма очередных цифр и переноса
            if (a[i] >= 10) {
               //случай,  когда  происходит  перенос  в
               //следующий разряд
                  r = 1;
                  a[i] -= 10;
            } else {
```

```
            //случай, когда переноса не происходит
                  r = 0;
            }
      }
      //если после сложения остался еще перенос, то
      //нужно добавить еще одну цифру
      if (r > 0) {
            a[0]++;
            a[a[0]] = r;
      }
}
```

Теперь необходимо рассмотреть деление длинного числа на короткое. Алгоритм:

```
int divide(thuge &a, int b) {
/*функция делит число a на короткое число b и
возвращает остаток от деления*/
      /*r - обозначает текущий остаток, к которому
      будет приписана очередная цифра*/
      int r = 0;
      //изначально остаток 0
      for(int i = a[0]; i >= 1; --i) {
      /*цикл от старших цифр к младшим*/
            r = r * base + a[i];
            //приписывание очередной цифры
            a[i] = r / b;
            //запись частного в результат
            r %= b;
            //пересчет остатка
      }
      /*Частное может содержать меньше цифр,
      поэтому нужно при необходимости уменьшить
      количество цифр*/
      while (a[0] > 1 && a[a[0]] == 0) {
            --a[0];
      }
      return r;
}
```

Итак, мы можем приступить к выполнению задания. Для начала зададим числа.

```
#define DMAX 30
typedef int thuge[DMAX];

thuge a, b, temp, result;
```

Введем числа, используя процедуру чтения больших чисел.

```
readHuge(a);
readHuge(b);
```

Теперь нам необходимо их перемножить

```
multiply(a, b, temp);
```

Полученный результат теперь необходимо поделить на 2 и сохранить. Это и будет ответом к задаче.

```
result = divide(temp, 2);
```

Осталось лишь вывести ответ с помощью модифицированной процедуры, определенной в задаче №1

```
writeHuge(result);
```

ЗАДАНИЕ ДЛЯ ДОМАШНЕЙ РАБОТЫ

Создать программу согласно полученному варианту. При выполнении домашней работы запрещается использовать сторонние классы и компоненты, реализующие заявленную функциональность.

ВАРИАНТЫ ЗАДАНИЙ

Условные обозначения:

(zn) – знак числа
N, N1 – величина порядка
E – основание числа
Запись числа (zn)0.m E N соответствует следующей записи числа $(zn)0.m*10^N$.

1. Смоделировать операцию деления действительного числа в форме (zn)m.n E N, где суммарная длина мантиссы (m+n) – до 30 значащих цифр, а величина порядка N – до 5 цифр на целое число длиной до 30 десятичных цифр. Результат выдать в форме (zn)0.m1 E N1.
2. Смоделировать операцию умножения двух действительных чисел в форме (zn)m.n E N, где суммарная длина мантиссы (m+n) – до 30 значащих цифр, а величина порядка N – до 5 цифр. Результат выдать в форме (zn)0.m1 E N1.
3. Смоделировать операцию деления целого числа длиной до 30 десятичных цифр на действительное число в форме (zn)m.n E N, где суммарная длина мантиссы (m+n) – до 30 значащих цифр, а величина порядка N – до 5 цифр. Результат выдать в форме (zn)0.m1 E N1.
4. Смоделировать операцию умножения действительного числа в форме (zn)m.n E N, где суммарная длина мантиссы (m+n) – до 30 значащих цифр, а величина порядка N – до 5 цифр на целое число длиной до 30 десятичных цифр. Результат выдать в форме (zn)0.m1 E N1.

5. Смоделировать операцию деления двух целых чисел длиной до 30 десятичных цифр каждое. Результат выдать в форме (zn)0.m E N, где длина мантиссы m – до 30 значащих цифр, а величина порядка N – до 5 цифр.

6. Смоделировать операцию умножения целого числа длиной до 30 десятичных цифр на действительное число в форме (zn)m.n E N, где суммарная длина мантиссы (m+n) – до 30 значащих цифр, а величина порядка N – до 5 цифр. Результат выдать в форме (zn)0.m1 E N1.

7. Смоделировать операцию деления двух действительных чисел в форме (zn)m.n E N, где суммарная длина мантиссы (m+n) – до 30 значащих цифр, а величина порядка N – до 5 цифр. Результат выдать в форме (zn)0.m1 E N1.

8. Смоделировать операцию сложения двух целых чисел длиной до 30 десятичных цифр каждое. Результат выдать в форме (zn)0.m E N, где длина мантиссы m – до 30 значащих цифр, а величина порядка N – до 5 цифр.

9. Смоделировать операцию сложения действительного числа в форме (zn)m.n E N, где суммарная длина мантиссы (m+n) – до 30 значащих цифр, а величина порядка N – до 5 цифр и целого числа длиной до 30 десятичных цифр. Результат выдать в форме (zn)0.m1 E N1

10. Смоделировать операцию сложения двух действительных чисел в форме (zn)m.n E N, где суммарная длина мантиссы (m+n) – до 30 значащих цифр, а величина порядка N – до 5 цифр. Результат выдать в форме (zn)0.m1 E N1.

11. Смоделировать операцию вычитания целого числа длиной до 30 десятичных цифр и действительного числа в форме (zn)m.n E N, где суммарная длина мантиссы (m+n) – до 30 значащих цифр, а величина порядка N – до 5 цифр. Результат выдать в форме (zn)0.m1 E N1.

12. Смоделировать операцию вычитания действительного числа в форме (zn)m.n E N, где суммарная длина мантиссы (m+n) – до 30 значащих цифр, а величина порядка N – до 5 цифр и целого числа длиной до 30 десятичных цифр. Результат выдать в форме (zn)0.m1 E N1.

13. Смоделировать операцию вычитания двух целых чисел длиной до 30 десятичных цифр каждое. Результат выдать в форме (zn)0.m E N, где длина мантиссы m – до 30 значащих цифр, а величина порядка N – до 5 цифр.

14. Смоделировать операцию вычитания целого числа длиной до 30 десятичных цифр и действительного числа в форме (zn)m.n E N, где суммарная длина мантиссы (m+n) – до 30 значащих цифр, а величина порядка N – до 5 цифр. Результат выдать в форме (zn)0.m1 E N1.

15. Смоделировать операцию вычитания двух действительных чисел в форме (zn)m.n E N, где суммарная длина мантиссы (m+n) – до 30 значащих цифр, а величина порядка N – до 5 цифр. Результат выдать в форме (zn)0.m1 E N1.

16. Смоделировать операцию умножения целого числа длиной до 30 десятичных цифр и действительного числа в форме (zn)0.m E N. Результат выдать в форме (zn)0.m1 E N1, где длина мантиссы m1 – до 30 значащих цифр, а величина порядка N1 – до 5 цифр.

17. Смоделировать операцию деления целого числа длиной до 30 десятичных цифр и действительного числа в форме (zn)0.m E N. Результат выдать в форме (zn)0.m1 E N1, где длина мантиссы m1 – до 30 значащих цифр, а величина порядка N1 – до 5 цифр.

18. Смоделировать операцию умножения целого числа длиной до 30 десятичных цифр и действительного числа в форме (zn)m.n E N, где суммарная длина мантиссы (m+n) – до 30 значащих цифр, а величина порядка N – до 5 цифр. Результат выдать в форме (zn)0.m1 E N1.

19. Смоделировать операцию деления целого числа длиной до 30 десятичных цифр на действительное число в форме (zn)m.n E N, где суммарная длина мантиссы (m+n) – до 30 значащих цифр, а величина порядка N – до 5 цифр. Результат выдать в форме (zn)0.m1 E N1.

20. Смоделировать операцию вычисления среднего арифметического целого числа длиной до 30 десятичных цифр и действительного числа в форме (zn)m.n E N, где суммарная длина мантиссы (m+n) – до 30 значащих цифр, а величина порядка N – до 5 цифр. Результат выдать в форме (zn)0.m1 E N1.

21. Смоделировать операцию вычисления среднего арифметического действительного числа в форме (zn)0.m E N и целого числа длиной до 30 десятичных цифр. Результат выдать в форме (zn)0.m1 E N1.

22. Смоделировать операцию вычисления среднего арифметического двух действительных чисел в форме (zn)m.n E N, где суммарная длина мантиссы (m+n) – до 30 значащих цифр, а величина порядка N – до 5 цифр. Результат выдать в форме (zn)0.m1 E N1.

23. Смоделировать операцию вычисления квадрата целого числа длиной до 30 десятичных цифр. Результат выдать в форме (zn)0.m E N, где длина мантиссы m – до 30 значащих цифр, а величина порядка N – до 5 цифр.

24. Смоделировать операцию вычисления квадрата действительного числа в форме (zn)m.n E N, где суммарная длина мантиссы (m+n) – до 30 значащих цифр, а величина порядка N – до 5 цифр. Результат выдать в форме (zn)0.m1 E N1.

25. Смоделировать операцию вычисления квадрата действительного числа в форме (zn)0.m E N. Результат выдать в форме (zn)0.m1 E N1.

26. Смоделировать операцию вычисления квадрата суммы двух действительных чисел в форме (zn)0.m E N. Результат выдать в форме (zn)0.m1 E N1.

27. Смоделировать операцию вычисления квадрата суммы двух целых чисел длиной до 30 десятичных цифр каждое. Результат выдать в форме (zn)0.m E N, где длина мантиссы m – до 30 значащих цифр, а величина порядка N – до 5 цифр.

28. Смоделировать операцию деления двух целых чисел длиной до 30 десятичных цифр каждое. Результат выдать в форме (zn)0.m E N, где длина мантиссы m – до 30 значащих цифр, а величина порядка N – до 5 цифр.

КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ

1. Изложите концепцию длинной арифметики.
2. Опишите как хранятся целые /вещественные числа в памяти.
3. Перечислите основные численные типы данных в C++.
4. Опишите способы хранения длинных чисел в памяти.
5. Объясните алгоритм сложения/вычитания длинных чисел.
6. Расскажите алгоритм умножения длинных чисел.
7. Раскройте алгоритм сравнения длинных чисел.
8. Сформулируйте концепцию алгоритма Евклида.
9. Расскажите, как осуществляется ввод и вывод длинных чисел.
10. Объясните алгоритм умножения/деления длинного числа на короткое.

ФОРМА ОТЧЕТА ПО ДОМАШНЕЙ РАБОТЕ

На выполнение домашней работы отводится 6 занятий (12 академических часа: 11 часов на выполнение и сдачу практического задания и 1 час на подготовку отчета).

Номер варианта студенту выдается преподавателем.

Отчет на защиту предоставляется в печатном виде.

Структура отчета (на отдельном листе(-ах)): титульный лист, формулировка задания, этапы выполнения работы, результаты выполнения, выводы.

ОСНОВНАЯ ЛИТЕРАТУРА

1. Алексеев В.Е. Графы и алгоритмы. Структуры данных. Модели вычислений [Электронный ресурс]/ В.Е. Алексеев, В.А. Таланов. — Электрон. текстовые данные. — М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 153 с. — Режим доступа: http://www.iprbookshop.ru/52186.html

2. Вирт Никлаус. Алгоритмы и структуры данных [Электронный ресурс]/ Никлаус Вирт— Электрон. текстовые данные. — Саратов: Профобразование, 2017. — 272 с.— Режим доступа: http://www.iprbookshop.ru/63821.html

3. Самуйлов С.В. Алгоритмы и структуры обработки данных [Электронный ресурс]: учебное пособие/ С.В. Самуйлов. — Электрон. текстовые данные. — Саратов: Вузовское образование, 2016. — 132 с.— Режим доступа: http://www.iprbookshop.ru/47275.html

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

4. Костюкова Н.И. Графы и их применение [Электронный ресурс]/ Н.И. Костюкова. — Электрон. текстовые данные. — М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 147 с. — Режим доступа: http://www.iprbookshop.ru/52185.html

5. Сундукова Т.О. Структуры и алгоритмы компьютерной обработки данных [Электронный ресурс]/ Т.О. Сундукова, Г.В. Ваныкина. — Электрон. текстовые данные. — М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 749 с.— Режим доступа: http://www.iprbookshop.ru/57384.html

Электронные ресурсы:

6. Научная электронная библиотека http://elibrary.ru
7. Электронно-библиотечная система «ЛАНЬ» http://e.lanbook.com
8. Электронно-библиотечная система «IPRbooks» http://www.iprbookshop.ru
9. Электронно-библиотечная система «Юрайт» http://www.biblio-online.ru
10. Электронно-библиотечная система «Университетская библиотека ONLINE» http://www.biblioclub.ru