**Лекция 2. Типы данных в языке С++**

**Цель лекции**: изучить классификацию типов и их внутреннее *представление* в языке С++, научиться работать со стандартными и *пользовательскими типами*.

Основная цель любой программы состоит в обработке каких-либо данных, например, чисел или текстов. Данные могут быть различного вида или типа и, в зависимости от этого, с ними можно выполнять разные действия.

В любом языке программирования каждая константа, *переменная*, результат *вычисления выражения* или функции должны иметь определенный *тип данных*.

**Тип данных** – это множество допустимых значений, которые может принимать тот или иной *объект*, а также множество допустимых операций, которые применимы к нему. В современном понимании тип также зависит от внутреннего представления информации.

Таким образом, данные различных типов хранятся и обрабатываются по-разному. *Тип данных* определяет:

* внутреннее представление данных в памяти компьютера;
* объем памяти, выделяемый под данные;
* множество (диапазон) значений, которые могут принимать величины этого типа;
* операции и функции, которые можно применять к данным этого типа.

Исходя из данных характеристик, необходимо определять тип каждой величины, используемой в программе для представления объектов. Обязательное описание типа позволяет компилятору производить *проверку допустимости* различных конструкций программы. От выбора типа величины зависит последовательность машинных команд, построенная компилятором.

### Классификация типов данных в С++

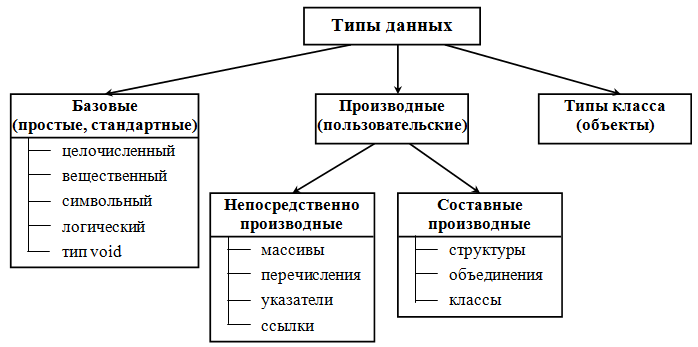
Современные языки программирования, как правило, могут иметь набор простых типов, являющихся встроенными в данный *язык программирования*, и средства для создания производных типов.

Объектно-ориентированные языки программирования позволяют определять типы класса.

Реализация простых типов данных заключается в способе представления значений данного типа в компьютере и в наборе операций, поддерживаемых для данного типа.

*Тип данных* определяет размер памяти, выделяемой под переменную данного типа при ее создании. *Язык программирования* C++ поддерживает следующие типы данных ([рис. 1.1](https://intuit.ru/studies/courses/648/504/lecture/11421?page=1#image.1.1)).

* *Базовые типы*. *Базовые типы* предопределены *стандартом языка*, указываются зарезервированными ключевыми словами и характеризуются одним значением. Их не надо определять и их нельзя разложить на более простые составляющие без потери сущности данных. *Базовые типы* объектов создают основу для построения более *сложных типов*.
* *Производные типы*. *Производные типы* задаются пользователем, и переменные этих типов создаются как с использованием базовых типов, так и типов классов.
* *Типы класса*. Экземпляры этих типов называются объектами.



**Рис. 1.1.**Типы данных в языке С++

Существует четыре *спецификатора* типа данных, уточняющих внутреннее *представление* и *диапазон* базовых типов:

|  |  |
| --- | --- |
| *short* (короткий) | длина |
| long (длинный) |
| *signed* (знаковый) | знак (модификатор) |
| *unsigned* (беззнаковый) |

Рассмотрим более подробно *базовые типы данных*.

### Целочисленный (целый) тип данных (тип int)

Переменные данного типа применяются для хранения целых чисел (*integer*). *Описание переменной*, имеющей тип int, сообщает компилятору, что он должен связать с идентификатором (именем) переменной количество памяти, достаточное для хранения целого числа во *время выполнения* программы.

Границы диапазона целых чисел, которые можно хранить в переменных типа int, зависят от конкретного компьютера, компилятора и операционной системы (от реализации). Для 16-разрядного процессора под него отводится 2 байта, для 32-разрядного – 4 байта.

Для внутреннего представления знаковых целых чисел характерно *определение* знака по старшему биту (0 – для положительных, 1 – для отрицательных). Поэтому число 0 во внутреннем представлении относится к положительным значениям. Следовательно, наблюдается *асимметрия* границ целых промежутков.

В *целочисленных типах* для всех значений определены следующий и предыдущий элементы. Для максимального следующим значением будет являться минимальное в этом же типе, предыдущее для минимального определяется как максимальное *значение*. То есть целочисленный *диапазон* условно можно представить сомкнутым в кольцо. Поэтому определены *операции* *декремента* для минимального и *инкремента* для максимального значений в *целых типах*.

От количества отводимой под *объект* памяти зависит множество допустимых значений, которые может принимать *объект*:

* *short* int – занимает 2 байта, следовательно, имеет диапазон от –32 768 до +32 767;
* int – занимает 4 байта, следовательно, имеет диапазон от –2 147 483 648 до +2 147 483 647;
* long int – занимает 4 байта, следовательно, имеет диапазон от –2 147 483 648 до +2 147 483 647;
* long long int – занимает 8 байтов, следовательно, имеет диапазон от –9 223 372 036 854 775 808 до +9 223 372 036 854 775 807.

Модификаторы *signed* и *unsigned* также влияют на множество допустимых значений, которые может принимать *объект*:

* *unsigned* *short* int – занимает 2 байта, следовательно, имеет диапазон от 0 до 65 535;
* *unsigned* int – занимает 4 байта, следовательно, имеет диапазон от 0 до 4 294 967 295;
* *unsigned* long int – занимает 4 байта, следовательно, имеет диапазон от 0 до 4 294 967 295;
* *unsigned* long long int – занимает 8 байтов, следовательно, имеет диапазон от 0 до 18 446 744 073 709 551 615.

Например:

unsigned int b;

signed int a;

int c;

unsigned d;

signed f;

Приведем несколько правил, касающихся записи целочисленных значений в исходном тексте программ.

* Нельзя пользоваться десятичной точкой. Значения 26 и 26.0 одинаковы, но 26.0 не является значением типа int.
* Нельзя пользоваться запятыми в качестве разделителей тысяч. Например, число 23,897 следует записывать как 23897.
* Целые значения не должны начинаться с незначащего нуля. Он применяется для обозначения восьмеричных или шестнадцатеричных чисел, так что компилятор будет рассматривать значение 011 как число 9 в *восьмеричной системе счисления*.

На практике рекомендуется использовать основной *целый тип*, то есть тип int. Данные основного *целого типа* практически всегда обрабатываются быстрее, чем данные других целых типов. Короткий тип *short* подойдет для хранения больших массивов чисел с целью экономии памяти при условии, что значения элементов не выходят за предельные границы для этих типов. Длинные типы необходимы в ситуации, когда не достаточно типа int.

### Вещественный (данные с плавающей точкой) тип данных (типы float и double)

Для хранения вещественных чисел применяются типы данных float (с одинарной точностью) и double (с двойной точностью). Смысл знаков "+" и "-" для вещественных типов совпадает с целыми. Последние незначащие нули справа от десятичной точки игнорируются. Поэтому варианты записи +523.5, 523.5 и 523.500 представляют одно и то же *значение*.

Для представления вещественных чисел используются два формата:

* с фиксированной точкой

[знак][целая часть].[дробная часть]

Например: –8.13; .168 (аналогично 0.168); 183. (аналогично 183.0).

* с плавающей точкой (экспоненциальной форме)

мантисса Е/е порядок

Например: 5.235e+02 (5.235 x 102 = 523.5); –3.4Е-03 (–3.4 x 10-03 = – 0.0034)

В большинстве случаев используется тип double, он обеспечивает более высокую *точность*, чем тип float. Максимальную *точность* и наибольший *диапазон* чисел достигается с помощью типа long double.

Величина с модификатором типа float занимает 4 байта. Из них 1 *бит* отводится для знака, 8 *бит* для избыточной *экспоненты* и 23 бита для *мантиссы*. Отметим, что старший *бит* *мантиссы* всегда равен 1, поэтому он не заполняется, в связи с этим *диапазон* модулей значений переменной с плавающей точкой приблизительно равен от 3.14E–38 до 3.14E+38.

Величина типа double занимает 8 байтов в памяти. Ее формат аналогичен формату float. Биты памяти распределяются следующим образом: 1 *бит* для знака, 11 *бит* для *экспоненты* и 52 бита для *мантиссы*. С учетом опущенного старшего бита *мантиссы* *диапазон* модулей значений переменной с двойной точностью равен от 1.7E–308 до 1.7E+308.

Величина типа long double аналогична типу double.

Например:

float a, b;

double x, y;

long double z;

### Символьный тип данных (тип char)

В стандарте C++ нет типа данных, который можно было бы считать действительно символьным. Для представления символьной информации есть два типа данных, пригодных для этой цели, – это типы char и wchar\_t.

*Переменная* типа char рассчитана на хранение только одного символа (например, буквы или пробела). В памяти компьютера символы хранятся в виде целых чисел. Соответствие между символами и их кодами определяется *таблицей кодировки*, которая зависит от компьютера и операционной системы. Почти во всех *таблицах кодировки* есть прописные и строчные буквы латинского алфавита, цифры 0, ..., 9, и некоторые специальные символы. Самой распространенной *таблицей кодировки* является *таблица символов* *ASCII* ( *American Standard Code for Information Interchange* – Американский стандартный код для обмена информацией).

Так как в памяти компьютера символы хранятся в виде целых чисел, то тип char на самом деле является подмножеством типа int.

Под величину символьного типа отводится 1 *байт*.

Тип char может использоваться со *спецификаторами* *signed* и *unsigned*. В данных типа *signed* char можно хранить значения в диапазоне от –128 до 127. При использовании типа *unsigned* char значения могут находиться в диапазоне от 0 до 255. Для кодировки используется код *ASCII*. Символы с кодами от 0 до 31 относятся к служебным и имеют самостоятельное *значение* только в операторах ввода-вывода.

Величины типа char также применяются для хранения чисел из указанных диапазонов.

Тип wchar\_t предназначен для работы с набором символов, для кодировки которых недостаточно 1 байта, например в кодировке *Unicode*. Размер типа wchar\_t равен 2 байтам. Если в программе необходимо использовать строковые *константы* типа wchar\_t, то их записывают с префиксом L, например, L "*Слово*".

Например:

char c='c';

char a,b;

char r[]={'A','B','C','D','E','F','\0'};

char s[] = "ABCDEF";

### Логический (булевый) тип данных (тип bool)

В языке С++ используется двоичная логика (*истина*, *ложь*). Лжи соответствует нулевое *значение*, истине – *единица*. Величины данного типа могут также принимать значения true и false.

Внутренняя форма представления значения false соответствует 0, любое другое *значение* интерпретируется как true. В некоторых компиляторах языка С++ нет данного типа, в этом случае используют тип int, который при истинных значениях выдает 1, а при ложных – 0. Под данные *логического типа* отводится 1 *байт*.

### Перечисляемый тип (тип enum)

Данный тип определяется как набор идентификаторов, являющихся обычными именованными целыми константами, которым приписаны уникальные и удобные для использования обозначения. Таким образом, перечисления представляют собой упорядоченные наборы целых значений. Они имеют своеобразный *синтаксис* и достаточно специфическую область использования.

*Переменная*, которая может принимать *значение* из некоторого списка определенных констант, называется *переменной перечисляемого типа* или *перечислением*. Данная *переменная* может принимать *значение* только из *именованных констант* списка. *Именованные константы* списка имеют тип int. Следовательно, *память*, соответствующая переменной перечисления, – это *память*, необходимая для размещения значения типа int.

Например:

enum year {winter, spring, summer, autumn};

enum week {Sunday, Monday, Tuesday, Wednesday,

Thursday, Friday, Saturday};

### Тип void

Множество значений этого типа пусто. Тип void имеет три назначения:

1. указание о невозвращении функцией значения;
2. указание о неполучении *параметров функцией*;
3. создание *нетипизированных указателей*.

Тип void в основном используется для *определения функций*, которые не возвращают значения, для указания пустого списка аргументов функции, как *базовый тип* для указателей и в *операции приведения* типов.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Основные типы данных | | | | | | |
| Тип | Обозначение | | Название | Размер памяти, байт (бит) | Диапазон значений |  |
| Имя типа | Другие имена |  |
| целый | int | *signed* | целый | 4 (32) | -2 147 483 648 до 2 147 483 647 |  |
| *signed* int |  |
| *unsigned* int | *unsigned* | беззнаковый целый | 4 (32) | 0 до 4 294 967 295 |  |
| *short* | *short* int | короткий целый | 2 (16) | -32 768 до 32 767 |  |
| *signed* *short* int |  |
| *unsigned* *short* | *unsigned* *short* int | беззнаковый короткий целый | 2 (16) | 0 до 65 535 |  |
| long | long int | длинный целый | 4 (32) | -2 147 483 648 до 2 147 483 647 |  |
| *signed* long int |  |
| *unsigned* long | *unsigned* long int | беззнаковый длинный целый | 4 (32) | 0 до 4 294 967 295 |  |
| long long | long long int | длинный-предлинный целый | 8 (64) | -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 |  |
| *signed* long long int |  |  |
| *unsigned* long long | *unsigned* long | беззнаковый длинный-предлинный целый | 8 (64) | 0 до 18 446 744 073 709 551 615 |  |
| long int |  |
| символьный | char | *signed* char | байт (целый длиной не менее 8 бит) | 1 (8) | -128 до 127 |  |
| *unsigned* char | - | беззнаковый байт | 1 (8) | 0 до 255 |  |
| wchar\_t | - | расширенный символьный | 2 (16) | 0 до 65 535 |  |
| вещественный | float | - | вещественный одинарной точности | 4 (32) | 3.4Е-38 до 3.4Е+38 (7 значащих цифр) |  |
| double | - | вещественный двойной точности | 8 (64) | 1.7Е-308 до 1.7Е+308 (15 значащих цифр) |  |
| long double | - | вещественный максимальной точности | 8 (64) | 1.7Е-308 до 1.7Е+308 (15 значащих цифр) |  |
|  | bool | - | логический | 1 (8) | true (1) или false (0) |  |
|  | *enum* | - | перечисляемый | 4 (32) | -2 147 483 648 до 2 147 483 647 |  |

### Преобразования типов

При *вычислении выражений* некоторые *операции* требуют, чтобы операнды имели соответствующий тип, в противном же случае на этапе компиляции выдается *сообщение об ошибке*. Например, операция взятия остатка от деления (%) требует целочисленных *операндов*. Поэтому в языке С++ есть возможность приведения значений одного типа к другому.

**Преобразование типов** – это приведение значения переменной одного типа в *значение* другого типа.

Выделяют *явное* и *неявное* *приведения типов*. При явном приведении указывается *тип переменной*, к которому необходимо преобразовать исходную переменную. При неявном приведении преобразование происходит автоматически, по правилам, заложенным в языке программирования С++.

Формат *операции* *явного преобразования* типов:

имя\_типа (операнд)

Например, int(x), float(2/5), long(x+y/0.5).

*Пример 1*.

//Взятие цифры разряда сотых в дробном числе

#include "stdafx.h"

#include <iostream>

using namespace std;

int \_tmain(int argc, \_TCHAR\* argv[]){

float s,t;

long int a,b;

printf("Введите вещественное число\n");

scanf("%f", &s);

t=s\*100;

a=(int)t;

//переменная t приводится к типу int в переменную a

b=a%10;

printf("\nЦифра разряда сотых числа %f равна %d.", s, b);

system("pause");

return 0;

}

Преобразования типов нужно применять с осторожностью, так как данная операция может приводить к *потере информации*. Например, после приведения длинного типа к более короткому происходит усечение информации из старших битов.

*Пример 2. Временной интервал*

Заданы моменты начала и конца некоторого промежутка времени в часах, минутах и секундах (в пределах одних суток). Найти продолжительность этого промежутка в тех же единицах.

Исходными данными для этой задачи являются шесть целых величин, задающих моменты начала и конца интервала, результатами – три целых величины (тип int ).

Обозначим переменные для хранения начала интервала hour1, min1 и sec1, для хранения конца интервала – hour2, min2 и sec2, а результирующие величины – *hour*, min и sec.

Для решения этой задачи необходимо преобразовать оба момента времени в секунды, вычесть первый из второго, а затем преобразовать результат обратно в часы, минуты и секунды. Следовательно, потребуется промежуточная *переменная* sum\_sec, в которой будет храниться величина интервала в секундах. Она может иметь весьма большие значения, ведь в сутках 86400 секунд, что выходит за пределы типа *short* int. Следовательно, для этой переменной выберем длинный *целый тип* ( long int, сокращенно long ).

Для перевода результата из секунд обратно в часы и минуты используется отбрасывание дробной части при делении целого числа на целое.

//Временной интервал. Форматированный ввод-вывод данных

#include "stdafx.h"

#include <iostream>

using namespace std;

int \_tmain(int argc, \_TCHAR\* argv[]){

int hour1, min1, sec1, hour2, min2, sec2, hour, min, sec;

long int sum\_sec;

printf("Введите время начала интервала (час мин сек)\n");

scanf("%d%d%d", &hour1,&min1,&sec1);

printf("Введите время окончания интервала (час мин сек)\n");

scanf("%d%d%d", &hour2,&min2,&sec2);

sum\_sec = (hour2-hour1)\*3600+(min2-min1)\*60+sec2-sec1;

hour = sum\_sec/3600;

min = (sum\_sec-hour\*3600)/60;

sec = sum\_sec-hour\*3600-min\*60;

printf("Продолжительность промежутка от %d:%d:%d до

%d:%d:%d\n",hour1,min1,sec1,hour2,min2,sec2);

printf(" равна %d:%d:%d\n",hour,min,sec);

system("pause");

return 0;

}

При выполнении математических операций производится неявное (*автоматическое) преобразование* типов, чтобы привести операнды выражений к общему типу или чтобы расширить короткие величины до размера целых величин, используемых в машинных командах. Выполнение преобразования зависит от специфики операций и от типа *операнда* или *операндов*.

*1. Преобразование целых типов со знаком*.

* *Целое со знаком преобразуется к более короткому целому со знаком*, с потерей информации: пропадают все разряды числа, которые находятся выше (или, соответственно – ниже) границы, определяющей максимальный размер переменной.
* *Целое со знаком преобразуется к более длинному целому со знаком*. Путем размножения знака. То есть все добавленные биты двоичного числа будут заняты тем же числом, которое находилось в *знаковом бите*: если число было положительным, то это будет, соответственно 0, если отрицательным, то 1.
* *Целое со знаком к целому без знака*. Первым шагом целое со знаком преобразуется к целому со знаком, соответствующему целевому типу, если этот тип данных крупнее. У получившегося значения бит знака не отбрасывается, а рассматривается равноправным по отношению к остальным битам, то есть теперь все биты образуют числовое значение.
* *Преобразование целого со знаком к плавающему типу* происходит без *потери информации*, за исключением случая преобразования типа long int или *unsigned* long int к типу float, когда точность часто может быть потеряна.

*2. Преобразование целых типов без знака*.

* *Целое без знака преобразуется к более короткому целому без знака или со знаком* путем усечения.
* *Целое без знака преобразуется к более длинному целому без знака или со знаком* путем добавления нулей слева.
* *Целое без знака преобразуется к целому со знаком того же размера*. Если взять для примера, *unsigned* *short* и *short* – числа в диапазоне от 32768 до 65535 превратятся в отрицательные.
* *Целое без знака преобразуется к плавающему типу*. Сначала оно преобразуется к значению типа *signed* long, которое затем преобразуется в плавающий тип.

*3. Преобразования плавающих типов*.

* *Величины типа* float *преобразуются к типу* double без изменения значения.
* *Величины* double *преобразуются к* float c некоторой *потерей точности*, то есть, количества знаков после запятой. Если значение слишком велико для float, то происходит потеря значимости, о чем сообщается во время выполнения.
* *При преобразовании величины с плавающей точкой к целым типам* она сначала преобразуется к типу long (дробная часть плавающей величины при этом отбрасывается), а затем величина типа long преобразуется к требуемому целому типу. Если значение слишком велико для long, то результат преобразования не определен. Обычно это означает, что на усмотрение компилятора может получиться любой "мусор". В реальной практике с такими преобразованиями обычно сталкиваться не приходится.

### Ключевые термины

**Базовые типы** – это типы данных, которые предопределены *стандартом языка*, указываются зарезервированными ключевыми словами, характеризуются одним значением и внутренним представлением.

**Вещественный тип** – это *базовый тип данных*, который применяется для хранения дробных чисел в формате с плавающей точкой.

**Логический (булевый) тип** – это *базовый тип данных*, который применяется для хранения значений двузначной логики.

**Неявное приведение типа** – это преобразование значения переменной к новому типу, которое происходит автоматически, по правилам, заложенным в языке программирования.

**Перечисляемый тип** – это *производный тип данных*, он определяется как набор идентификаторов, являющихся именованными целыми константами, которым приписаны уникальные обозначения

**Преобразование типов** – это приведение значения переменной одного типа в *значение* другого типа.

**Производные типы** – это типы данных, которые задаются пользователем.

**Символьный тип** – это *базовый тип данных*, который применяется для хранения символов или *управляющих последовательностей* в виде кода.

**Тип данных** – это множество допустимых значений, которые может принимать тот или иной *объект*, а также множество допустимых операций, которые применимы к нему.

**Типы класса** – это типы данных, экземплярами которых являются объекты.

**Целочисленный тип** – это *базовый тип данных*, который применяется для хранения целых чисел.

**Явное приведение типа** – это преобразование значения переменной к новому типу, при котором указывается *тип переменной*, к которому необходимо привести исходную переменную.

### Краткие итоги

1. Для организации хранения данных и корректного выполнения операций над ними в языках программирования определены типы данных.
2. Типы характеризуются схожим внутренним представлением данных в памяти компьютера; объемом памяти, выделяемым под данные; множеством (диапазоном) принимаемых значений; допустимыми операциями и функциями.
3. В языке С++ типы классифицируются на базовые, производные и классы.
4. Для базовых типов определены их подмножества и расширения, что обеспечивает повышение точности расчетов или экономный расход памяти.
5. Над типами данных определена операция преобразования типов. Ее следует применять с осторожностью при переходе к типу, у которого меньше по модулю границы диапазонов.

### Лабораторная работа 1. Типы данных в языке С++

**Цель работы**: изучить классификацию типов и их внутреннее *представление* в языке С++, научиться работать со стандартными и *пользовательскими типами*.

При выполнении лабораторной работы для каждого задания требуется написать программу на языке С++, которая получает на входе числовые данные, выполняет их обработку в соответствии с требованиями задания и выводит результат на экран. Ввод данных осуществляется с клавиатуры с учетом требований к входным данным, содержащихся в постановке задачи (к задачам 2 и 4 приведен рекомендуемый вид диалога во время работы программы). Ограничениями на *входные данные* является допустимый *диапазон* значений используемых *числовых типов* в языке С++.

**Теоретические сведения**.

Ознакомьтесь с материалом лекции 1.

**Задания к лабораторной работе**.

Выполните приведенные ниже задания.

1. Найдите сумму первых трех цифр дробной части *вещественного числа*. Например, для числа 23,16809 она равна 15.
2. Составьте программу вычисления стоимости поездки на автомобиле на дачу (туда и обратно). Исходными данными являются: расстояние до дачи (в километрах); количество бензина, которое потребляет автомобиль на 100 км пробега; цена одного литра бензина. Ниже представлен рекомендуемый вид диалога во время работы программы.
3. Вычисление стоимости поездки на дачу.
4. Расстояние до дачи (км) – 67
5. Расход бензина (л на 100 км) – 8.5
6. Цена литра бензина (руб.) – 23.7

Поездка на дачу обойдется в 269 руб. 94 коп.

1. Составьте *линейную программу*, печатающую значение 1, если указанное *высказывание* является истинным, и 0 – в противном случае. Величина d является корнем только одного из уравнений ax2 + bx + c = 0 и *mx* + n = 0 относительно х.
2. Составьте программу, которая преобразует введенное с клавиатуры дробное число в денежный формат. Например, число 12,348 должно быть преобразовано к виду 12 руб. 35 коп. Ниже представлен рекомендуемый вид диалога во время работы программы. Данные, вводимые пользователем, выделены жирным шрифтом.
3. Преобразование числа в денежный формат.
4. Введите дробное число – 23,6

23.6 руб. – это 23 руб. 60 коп.

**Указания к выполнению работы**.

Каждое задание необходимо решить в соответствии с изученными методами обработки данных и преобразования типов данных в языке С++.

Следует реализовать каждое задание в соответствии с приведенными этапами:

* изучить словесную постановку задачи, выделив при этом все виды данных;
* сформулировать математическую постановку задачи;
* выбрать метод решения задачи, если это необходимо;
* разработать графическую схему алгоритма;
* записать разработанный алгоритм на языке С++;
* разработать контрольный тест к программе;
* отладить программу;
* представить отчет по работе.

**Требования к отчету**.

Отчет по лабораторной работе должен соответствовать следующей структуре.

* Титульный лист.
* Словесная постановка задачи. В этом подразделе проводится полное описание задачи. Описывается суть задачи, анализ входящих в нее физических величин, область их допустимых значений, единицы их измерения, возможные ограничения, анализ условий при которых задача имеет решение (не имеет решения), анализ ожидаемых результатов.
* Математическая модель. В этом подразделе вводятся математические описания физических величин и математическое описание их взаимодействий. Цель подраздела – представить решаемую задачу в математической формулировке.
* Алгоритм решения задачи. В подразделе описывается разработка структуры алгоритма, обосновывается абстракция данных, задача разбивается на подзадачи. Схема алгоритма выполняется по ЕСПД (ГОСТ 19.003-80 и ГОСТ 19.002-80).
* Листинг программы. Подраздел должен содержать текст программы на языке программирования С++, реализованный в среде MS Visual Studio 2010.
* Контрольный тест. Подраздел содержит наборы исходных данных и полученные в ходе выполнения программы результаты.
* Выводы по лабораторной работе.
* Ответы на контрольные вопросы.

**Контрольные вопросы**

1. Почему в языке С++ определена строгая типизация данных, используемых в программе?
2. Как определяются границы диапазона базового типа в зависимости от выделяемой под этот тип памяти?
3. С какой целью в С++ определен тип void?
4. Какой объем памяти выделяется под переменную типа void? Какие значения может принимать переменная типа void?
5. Почему наблюдается *асимметрия* значений границ диапазонов целочисленных типов?
6. Чему будет равно значение операции *инкремента* для максимального числа в целочисленном типе? А каков результат *декремента* для минимального значения в таком же типе?
7. Почему запись целых чисел нельзя начинать с незначащих нулей?
8. Каким образом представлено число ноль в вещественных типах?
9. Почему в С++ *символьный тип* считается подмножеством целочисленного типа?
10. Каким образом можно инициализировать переменную перечисляемого типа?
11. При преобразовании целого со знаком к целому без знака всегда ли будет получено исходное числовое значение? Ответ обоснуйте.