Основные понятия языка программирования C++

Простейшая программа на С++

Самая короткая программа на языке Си++:

```
// Простейшая программа
int main() { return 1; }
```

1-ая строчка – комментарий, служит для пояснения.

Признак комментария: // или /* … */.

main – имя главной функции программы. С функции main всегда начинается выполнение.

У функции есть имя (main), после имени в круглых скобках перечисляются аргументы или параметры функции (в данном случае у функции main аргументов нет).

У функции может быть результат или возвращаемое значение. Если функция не возвращает никакого значения, то это обозначается ключевым словом void.

В фигурных скобках записывается тело функции – действия, которые она выполняет.

Оператор return 1 означает, что функция возвращает результат – целое число 1.

Если говорим об объектно-ориентированной программе, то она должна создать объект какого-либо класса и послать ему сообщение. Чтобы не усложнять программу, мы воспользуемся одним из готовых, предопределенных классов – классом ostream (поток ввода-вывода, базовый класс для iostream). Этот класс определен в *файле* заголовков "iostream.h". Поэтому надо включить *файл* заголовков в программу:

```
#include <iostream.h>
int main() { return 1; }
```

Кроме класса, *файл* заголовков определяет глобальный объект этого класса cout. Объект называется глобальным, поскольку доступ к нему возможен из любой части программы.

Этот объект выполняет вывод на консоль.

В функции main мы можем к нему обратиться и послать ему сообщение:

```
#include <iostream.h>
int main()
{
   cout << "Hello, world!" << endl;
   return 1;}
```

Операция сдвига << для класса ostream определена как "вывести".
Таким образом, программа посылает объекту cout сообщения "вывести строку Hello, world!" и "вывести перевод строки" (endl обозначает новую строку).
В ответ на эти сообщения объект cout выведет строку "Hello, world!" на консоль и переведет курсор на следующую строку.

Имена, переменные, константы

Для символического обозначения величин, имен функций и т.п. используются *имена* или *идентификаторы*.
Идентификаторы в языке Си++ – это последовательность знаков, начинающаяся с буквы или знака подчеркивания.
В *идентификаторах* можно использовать заглавные и строчные латинские буквы, цифры и знак подчеркивания.
Длина *идентификаторов* произвольная.
Примеры правильных *идентификаторов*:
abc A12 NameOfPerson BYTES_PER_WORD
abc и Abc – два разных *идентификатора*, т.е. заглавные и строчные буквы различаются.

Пример неправильного *идентификатора*: 12X a-b

Ряд слов в языке Си++ не может использоваться в качестве *идентификаторов*.
Такие зарезервированные слова называются ключевыми.
Примеры:

asm auto bad_cast bad_typeid bool break
case catch char clas cons continue default
delete do double else enum
extern float for friend goto if
inline int long mutable namespace
new operator private protected public
register return short signed sizeofstatic static_cast
struct switch template
thenthis throw try type_info
typedef typeidunion unsigned using

virtual void volatile while xalloc

Пример:

```
int max(int x, int y) {
    if (x > y)
        return x;
    else
       return y;}
```

max, x и y – *имена* или *идентификаторы*.
Слова int, if, return и else – ключевые слова, они не могут быть *именами переменных* или функций и используются для других целей.

Переменная – это символическое обозначение величины в программе. Значение *переменной* во время выполнения программы может изменяться.

С точки зрения архитектуры компьютера, ***переменная*** – это символическое обозначение ячейки оперативной памяти программы, в которой хранятся данные. Содержимое этой ячейки – это текущее значение *переменной*.

В Си++ переменную перед использованием необходимо **объявить**.

Объявить *переменную* с *именем* x можно так: int x;

В объявлении первым стоит название типа *переменной* int (целое число), а затем *идентификатор* x – *имя переменной*.

У *переменной* x есть тип – в данном случае целое число.

Тип переменной определяет, какие возможные значения эта переменная может принимать и какие операции можно выполнять над данной переменной.

Тип *переменной* изменить нельзя, т.е. пока *переменная* x существует, она всегда будет целого типа.

Язык Си++ – это строго *типизированный язык*.
Любая величина, используемая в программе, принадлежит к какому-либо типу. При любом использовании *переменных* в программе проверяется, применимо ли выражение или операция к типу *переменной*.
Часто смысл выражения зависит от типа участвующих в нем *переменных*.
Например, если мы запишем x+y, где x – объявленная выше *переменная*, то *переменная* y должна быть одного из числовых типов.
Соответствие типов проверяется во время компиляции программы. Если компилятор обнаруживает несоответствие типа *переменной* и ее использования, он выдаст ошибку (или предупреждение).
Во время выполнения программы типы не проверяются. Это позволяет обнаружить и исправить большое количество ошибок на стадии компиляции и не замедляет выполнения

программы.

Переменной можно присвоить какое-либо значение с помощью операции **присваивания**.
Присвоить – это значит установить текущее значение *переменной*.
Или: операция присваивания запоминает новое значение в ячейке памяти, которая обозначена *переменной*.

```
int x;          // объявить целую переменную x
int y;          // объявить целую переменную y
x = 0;          // присвоить x значение 0
y = x + 1;      // присвоить y значение x + 1, т.е. 1
x = 1;          // присвоить x значение 1

y = x + 1;  // присвоить y значение x + 1,  теперь уже 2
```

Константы – это неизменяемая величина

Явная запись значения в программе – это константа.

Не всегда удобно записывать константы в тексте программы явно. Чаще используются *символические константы*.

Например, если запишем : const int BITS_IN_WORD = 32;

то затем *имя* BITS_IN_WORD можно будет использовать вместо целого числа 32.

Преимущества такого подхода.

1) *имя* BITS_IN_WORD (битов в машинном слове) дает хорошую подсказку, для чего используется данное число.

Без комментариев понятно, что выражение

b / BITS_IN_WORD (значение b разделить на число 32) вычисляет количество машинных слов, необходимых для хранения b битов информации.

2) Если по каким-либо причинам надо изменить эту константу, потребуется изменить только одно место в программе – определение константы, оставив все случаи ее использования как есть.

Выражения

Программа оперирует с данными. Числа можно +, -, /, *. Из разных величин можно составлять *выражения*, результат вычисления которых – новая величина.

Примеры *выражений*:

X * 12 + Y // значение X *на 12 и к результату + значение Y

val < 3 // сравнить значение val с 3

-9 // константное выражение -9

Выражение, после которого стоит точка с запятой – это оператор-выражение. Его смысл состоит в том, что компьютер должен вычислить *выражение*.

x + y – 12; // сложить значения x и y и затем вычесть

12a = b + 1; // прибавить единицу к значению b и запомнить результат в переменной a

Выражения – это переменные, функции и константы, называемые операндами, объединенные знаками операций.

Операции могут быть ***унарными*** – с одним операндом, например, минус; могут быть *бинарные* – с двумя операндами, например *сложение* или *деление*.

В Си++ есть операция с тремя операндами – условное *выражение*.

В типизированном языке ся Си++, у переменных и констант есть определенный тип.

Есть он и у результата *выражения*. Например, операции *сложения* (+), *умножения* (*), *вычитания* (-) и *деления* (/), примененные к целым числам, выполняются по общепринятым математическим правилам и дают в результате целое значение. Те же операции можно применить к вещественным числам и получить вещественное значение.

Операции ***сравнения***: больше (>), меньше (<), равно (==), не равно (!=) сравнивают значения чисел и выдают логическое значение: истина (true) или ложь (false).

Типы данных в C++. В любом ЯП каждая константа, *переменная*, результат *вычисления выражения* или функции должны иметь *тип данных*.

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

Тип также зависит от внутреннего представления информации.

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.

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.

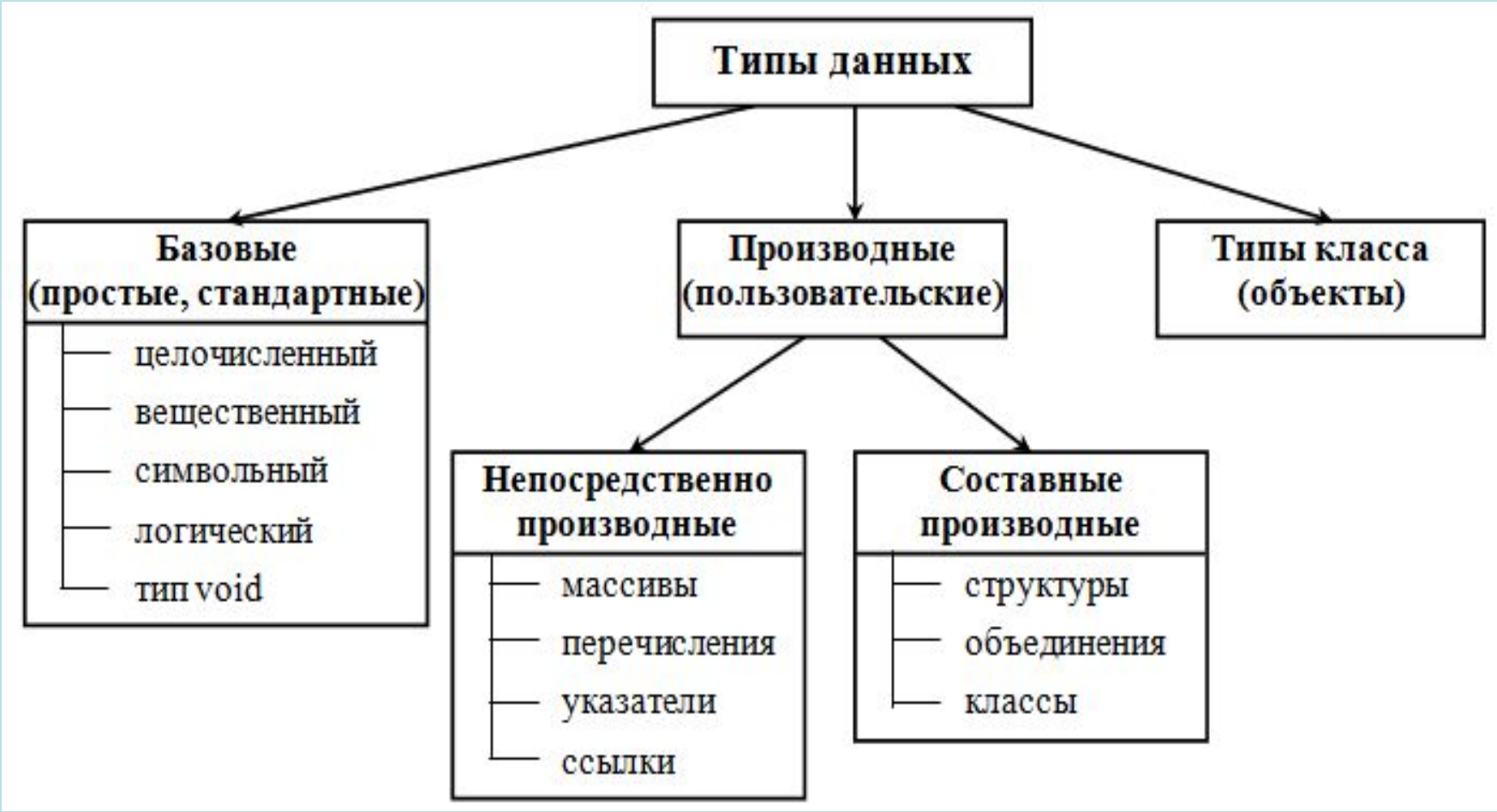
ЯП C++ поддерживает следующие типы данных.

- *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.

Типы данных в C++:



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

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

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, имеет диапазон от –32 768 до +32 767;
- int – занимает 4 байта, имеет диапазон от –2 147 483 648 до +2 147 483 647;
- long int – занимает 4 байта, имеет диапазон от –2 147 483 648 до +2 147 483 647;
- long long int – занимает 8 байтов, 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, имеет диапазон от 0 до 65 535;
- *unsigned* int – занимает 4 байта, имеет диапазон от 0 до 4 294 967 295;
- *unsigned* long int – занимает 4 байта, имеет диапазон от 0 до 4 294 967 295;
- *unsigned* long long int – занимает 8 байтов, имеет диапазон от 0 до 18 446 744 073 709 551 615.

Примеры:

- unsigned int b; signed int a;
- int c; unsigned d;
- signed f;

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. 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 E/e порядок

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

- Тип double 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.

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 C++ 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.

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.
- 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:
- указание о невозвращении функцией значения;
- указание о неполучении *параметров функцией*;
- 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.

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*.

В ЯП C++ 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 C++.

Формат *операции явного преобразования* типов:

- имя_типа (операнд)

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

Пример.

```
//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;
}
```

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 ниже) границы, определяющей максимальный размер переменной.
- ***Целое со знаком преобразуется к более длинному целому со знаком.*** Путем размножения знака. 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*.** Сначала целое со знаком преобразуется к целому со знаком, соответствующему целевому типу, если этот тип данных крупнее. У получившегося значения бит знака не отбрасывается, 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 с 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.

Ключевые термины

- **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

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

Пример 1. Пусть необходимо разделить целых числа.

```
int x,y;
x=15;
Y=2;
```

Делим: x/y=7.

Если сделать преобразования, например: x/y.=7.5

Тогда число 2 является вещественным, значит и результат будет вещественный.

Само число 2 с точки зрения математики не изменилось, ведь 2=2.0.

Этот же прием можно было применить к 15, результат был бы тем же, а можно было сразу к двум числам, но зачем ,если хватает одного из двух.

Пример 2.

```
int i1 = 11;
int i2 = 3;
float x = (float)i1 / i2;
```

Стандартная библиотека ввода-вывода Си (заголовочный файл stdio.h) Функция консольного форматного вывода

Вызов функции:

printf(*форматная_строка*, *список_вывода*);

В каком виде выводим. Форматная строка – это строка выводимого текста, в который вставлены **спецификации форматов** для выводимых значений.

Что выводим: переменные, простые и с индексами, константы, произвольные выражения; перечисление через запятую.

Спецификации форматов

Общий вид спецификации формата:

%*ширина_поля*.*точность* *спецификатор*

Число символов (позиций). занимаемых значением. Если ширина недостаточна, то она игнорируется; если избыточна, то значение дополняется пробелами.

Число цифр дробной части – только для вещественных чисел.

Какое значение выводим

Основные спецификаторы

- d - для целых значений (типов int, char, unsigned);
- f - для вещественных значений (типов float, double);
- e - для вещественных значений (типов float, double)с указанием порядка, т. е. в виде:
  *знак_числа*m.dddd**e***знак_порядка*ххх,
  где m.dddd - изображение мантиссы числа; m - одна десятичная цифра; dddd - последовательность десятичных цифр; **e** - признак порядка (десятичного); ххх - десятичные цифры для представления порядка числа.
- u – для значений без знака;
- c - для одиночного символа (типов int, char, unsigned);
- s - для строк.

Простейший пример форматного вывода

форматная строка может содержать управляющие символы.

```
float s; int a,b;...
printf("a=%2d     b=%4d\n     s=%4.1\n", a,b,s);
```

Форма вывода

```
a=<a>
b=<b>
s=<s>
```

Если a=-2, b=93, s=3.22,
то на экране получим:

```
a=-2      b=
   93
s= 3.2
```

Функция консольного форматного ввода

Вызов функции:

scanf(*форматная_строка*, *список_ввода*);

В каком виде вводимые значения представлены в консольном экране.

Что вводить: **адреса** вводимых **переменных** через запятую.

Почему адреса? Почему только переменных?

Форматная строка при вводе содержит только спецификации форматов. текста не содержит.

Почему?

Вводимые значения могут разделяться пробелами (одним или несколькими) или переводом строки (нажатием клавиши *Enter*), после последнего введенного значения надо обязательно нажать *Enter.*

Простейший пример форматного ввода

```
int i; float a;
printf("Введите i и a\n"); /* вывод приглашения к вводу */
scanf("%d%f", &i, &a);... /* спецификации могут */
                          /* не содержать ширины и точности */
```

Форма ввода

Введите i и a

$$\left\{ \begin{array}{c} <i> \quad <a> \\ \hline <i> \\ <a> \end{array} \right\}$$

{ } – альтернативный вывод

Замечание. При вводе строк символов с помощью функции ***scanf()*** (и с помощью ***cin***) действуют более сложные правила. Так, в буфер устройства ввода считываются все символы до нажатия *Enter,* а в вводимую строковую переменную передаются символы до первого пробела. Такой принцип работы scanf() имеет свои преимущества, но они слишком трудны при начальном освоении языка Си. Поэтому для ввода и вывода строк лучше пользоваться функциями ***gets()*** и ***puts()*** из стандартной библиотеки Си.