# Numbering System:

A numbering system is a system for writing numerical values in terms of symbols. In a numbering system certain number of distinct values are recognized and then each value is assigned a distinct symbol. Based upon the number of distinct values it recognizes and the corresponding set of symbols, it gets its name. Thus, if a numbering system recognizes two values (and corresponding two symbols) then it is known as binary numbering system. Similarly, a numbering system that recognizes ten distinct values (and corresponding ten symbols) is known as decimal numbering system.

There are four numbering systems viz. binary, decimal, octal and hexadecimal which are commonly used in computing industry. There details are as follows:

## Binary: Octal: Hexadecimal:

|  |  |
| --- | --- |
| **Symbol** | **Value** |
| 0 | Zero |
| 1 | One |

|  |  |
| --- | --- |
| **Symbol** | **Value** |
| 0 | Zero |
| 1 | One |
| 2 | Two |
| 3 | Three |
| 4 | Four |
| 5 | Five |
| 6 | Six |
| 7 | Seven |

|  |  |
| --- | --- |
| **Symbol** | **Value** |
| 0 | Zero |
| 1 | One |
| 2 | Two |
| 3 | Three |
| 4 | Four |
| 5 | Five |
| 6 | Six |
| 7 | Seven |
| 8 | Eight |
| 9 | Nine |
| A | Ten |
| B | Eleven |
| C | Twelve |
| D | Thirteen |
| E | Fourteen |
| F | Fifteen |

## Decimal:

|  |  |
| --- | --- |
| **Symbol** | **Value** |
| 0 | Zero |
| 1 | One |
| 2 | Two |
| 3 | Three |
| 4 | Four |
| 5 | Five |
| 6 | Six |
| 7 | Seven |
| 8 | Eight |
| 9 | Nine |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Symbol** | **Value** | **Binary** | **Decimal** | **Octal** | **Hex** |
| 0 | Zero | 0000 | 0 | 0 | 0 |
| 1 | One | 0001 | 1 | 1 | 1 |
| 2 | Two | 0010 | 2 | 2 | 2 |
| 3 | Three | 0011 | 3 | 3 | 3 |
| 4 | Four | 0100 | 4 | 4 | 4 |
| 5 | Five | 0101 | 5 | 5 | 5 |
| 6 | Six | 0110 | 6 | 6 | 6 |
| 7 | Seven | 0111 | 7 | 7 | 7 |
| 8 | Eight | 1000 | 8 |  | 8 |
| 9 | Nine | 1001 | 9 |  | 9 |
| A | Ten | 1010 |  |  | A |
| B | Eleven | 1011 |  |  | B |
| C | Twelve | 1100 |  |  | C |
| D | Thirteen | 1101 |  |  | D |
| E | Fourteen | 1110 |  |  | E |
| F | Fifteen | 1111 |  |  | F |

# How to represent those numerical values for which there is no direct symbolic representation in a given numbering system?

Let's take a number one hundred and sixty nine. In decimal numbering system, we do not have any particular symbol which recognizes this number single handedly on the paper. To represent such number we make use of place-value concept. By this concept a number may comprise of several places and a same or different digit of respective numbering system may fill up that place. The value of the digit however will be according to its place in the sequence. For ex. in decimal numbering system we would represent one hundred and sixty nine as 169. It has three places. The value of 1 in 169 is not one but is one hundred and that of 6 is not six but sixty. We use this technique to represent those values for which there is no direct symbol available in respective numbering system.

# Conversion from Decimal Numbering System to Binary Numbering System:

# Conversion from Binary Numbering System to Hexadecimal Numbering System:

# How real world data values could be represented using digits of binary numbering system?

In real world, we typically come across two kinds of values, simple values and complex values. Following table lists typical simple values with their types.

|  |  |
| --- | --- |
| **Simple values** | **Type of data** |
| 'A' wing of an apartment or 'C' grade in a mark sheet. | Character |
| Count of things | Integer |
| Price, Weight, Average | Decimal |
| Yes/No, True/False | Boolean |

Complex values can be constructed using simple values. Examples of complex values include a two or three dimensional point, a complex number, a date or a vector. One can think unlimited number of complex values.

The question now is, how binary digits can help us represent simple values? If we succeed in finding representation for simple values then it's as good as we found representation for complex values (because complex values can be thought of as made up of simple values).

## Representing characters using binary digits.

Refer to following table which maps characters to binary numbers.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Char** | **Binary** | **Char** | **Binary** | **Char** | **Binary** | **Char** | **Binary** |
| ! | 00100001 | : | 00111010 | S | 01010011 | l | 01101100 |
| " | 00100010 | ; | 00111011 | T | 01010100 | m | 01101101 |
| # | 00100011 | < | 00111100 | U | 01010101 | n | 01101110 |
| $ | 00100100 | = | 00111101 | V | 01010110 | o | 01101111 |
| % | 00100101 | > | 00111110 | W | 01010111 | p | 01110000 |
| & | 00100110 | ? | 00111111 | X | 01011000 | q | 01110001 |
| ' | 00100111 | @ | 01000000 | Y | 01011001 | r | 01110010 |
| ( | 00101000 | A | 01000001 | Z | 01011010 | s | 01110011 |
| ) | 00101001 | B | 01000010 | [ | 01011011 | t | 01110100 |
| \* | 00101010 | C | 01000011 | \ | 01011100 | u | 01110101 |
| + | 00101011 | D | 01000100 | ] | 01011101 | v | 01110110 |
| , | 00101100 | E | 01000101 | ^ | 01011110 | w | 01110111 |
| - | 00101101 | F | 01000110 | \_ | 01011111 | x | 01111000 |
| . | 00101110 | G | 01000111 | ` | 01100000 | y | 01111001 |
| / | 00101111 | H | 01001000 | a | 01100001 | z | 01111010 |
| 0 | 00110000 | I | 01001001 | b | 01100010 | { | 01111011 |
| 1 | 00110001 | J | 01001010 | c | 01100011 | | | 01111100 |
| 2 | 00110010 | K | 01001011 | d | 01100100 | } | 01111101 |
| 3 | 00110011 | L | 01001100 | e | 01100101 | ~ | 01111110 |
| 4 | 00110100 | M | 01001101 | f | 01100110 |  |  |
| 5 | 00110101 | N | 01001110 | g | 01100111 |  |  |
| 6 | 00110110 | O | 01001111 | h | 01101000 |  |  |
| 7 | 00110111 | P | 01010000 | i | 01101001 |  |  |
| 8 | 00111000 | Q | 01010001 | j | 01101010 |  |  |
| 9 | 00111001 | R | 01010010 | k | 01101011 |  |  |

This is how keyboard communicates with a computer system. When you press a key on a keyboard say 'A' key, it's not graphics of 'A' that is communicated to the computer system but a binary code which recognizes the key being pressed as 'A' key is communicated to computer system.

A word or a sentence can be thought of as a complex data when compared against alphabets. Now if we have above character code system in place then it's easy to represent a word or a sentence using binary digits.

For ex. we can represent word HI as 01001000 01001001.

## Representing integers using binary digits.

Refer to topic "Conversion from Decimal Numbering System to Binary Numbering System". In that, we have already seen how a binary number can represent a decimal number.

# Representing fractions using binary digits.

Using some complex scheme it's possible to represent fractions in binary numbering system.

# Representing Boolean using binary digits.

A single binary digit 0 can be used to represent false value and a binary digit 1 can be used to represent true value. Though single binary digits are enough to represent Boolean values, its common to represent false as 00000000, a group of 8 binary digits and true as 00000001 again a group of eight binary digits.

Thus we observed, it's possible to represent real world data using digits of binary numbering system.

# What is a bit?

It's a short form of **Bi**nary Digi**t**. It represents a place which can either be filled with 0 or 1.

# What is Byte?

A group of 8 bits is known as a byte. Byte is used as a unit of measurement for measuring capacity of a memory.

# What is Nibble?

A group of 4 bits is known as a nibble.

# What are KB, MB and GB?

KB stands for Kilo Bytes, MB stands for Mega Bytes and GB stands for Giga Bytes. These are all units of measurement for measuring quantity of bytes. There are other higher UOMs also available, but they are out of scope of this document.

The relationship between byte and these UOMs goes as below:

1KB = 1024 Bytes.

1 MB = 1024 KB = 10, 48,576 Bytes (≈10 Million)

1 GB = 1024 MB = 1,07,37,41,824 Bytes (≈1 Billion)

# What are major building blocks of a computer system?



Power Supply

Disk Drives

Motherboard

IO Ports

Memory

Microprocessor

Registers

ALU

CU



Data Bus



Control Bus

Address Bus

## About Microprocessor:

Think microprocessor as a library of hardware circuits dedicated for performing arithmetic operations such as addition, multiplication, division etc. and logical operations such as 'and' and 'or'.

CU which stands for Control Unit, selects particular hardware circuit(s) based upon operation code it receives.

ALU stands for Arithmetic Logical Unit. It is this part of a microprocessor where arithmetic and/or logical operations are performed.

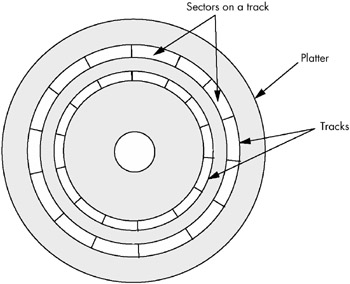
Registers are temporary memory locations which are present inside microprocessor. Data before it is fed to ALU is first stored in Registers. Often outcome of ALU is stored in registers.

IO Ports are connectors which usually popup on the front and backside of CPU which are used to plug in devices such as monitor, keyboard, mouse, USB drives etc.

Memory (registers) inside microprocessor is very limited, not enough to store significant amount of data. We therefore need external source of memory which provides sufficient storage to store significant amount of data. We can broadly classify memory into two types viz. primary memory and secondary memory. Primary memory is located on motherboard (RAM) and secondary memories such as disk drives are located in bays provided inside CPU cabinet.

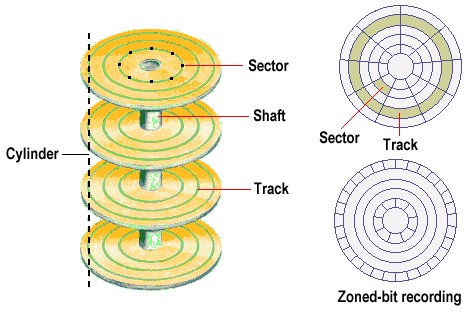
Disk Drives are permanent memories, which means that data is retained in these memories even if computer system is switched off. Disk drives often store data magnetically or optically.

## Construction of Hard Disk Drives

A hard disk drive platter (or disk) is the circular disk on which magnetic data is stored in a hard disk drive. The rigid nature of the platters in a hard drive is what gives them their name (as opposed to the flexible materials which are used to make floppy disks). Hard drives typically have several platters which are mounted on the same spindle. A platter can store information on both sides, requiring two heads per platter.

The magnetic surface is logically divided into concentric circles known as tracks. A track is further divided into sectors. Each sector typically holds 512 bytes of information.

Number of sectors come together to form logically high level entity known as File. File Allocation Table commonly known as FAT is a table that contains information of which sectors make up a particular file and in which order they should be read or written.



A directory is an organizational unit used to organize files into hierarchical structure. A directory besides files may contain sub-directories which may contain further sub-sub-directories.

When a hard disk is formatted, OS installs an apex level directory known as root directory. Root directory is often denoted by backward slash "\" immediately after the drive letter.

Study material: https://www.bbc.co.uk/bitesize/guides/zhppfcw/revision/1

Functions of Operating System:

<https://www.geeksforgeeks.org/functions-of-operating-system/> clear

Command shell commands to learn: cls, set, echo, md, tree, cd, rd, dir, copy, type, del, ren

# Concept of assembler, compiler, interpreter, loader and linker

<https://www.efaculty.in/c-language/concept-of-assembler-compiler-interpreter-loader-and-linker/>

<https://www.tenouk.com/ModuleW.html>

<https://www.bbc.co.uk/bitesize/guides/zmthsrd/revision/1>

# Character set

All C/C++ programs are made out of following character set.

* 0, 1, … , 9
* A, B, … , Z
* a, b, … , z
* ! ” # $ % & ‘ ( ) \* + , - . / : ; < = > ? @ [ \ ] ^ \_ { | } ~
* \a \b \f \n \r \t \v

# Preprocessor directives

Preprocessor is a separate program invoked by the compiler as the first part of translation. Refer <http://www.cplusplus.com/doc/tutorial/preprocessor/> for more detailed description.

## Preprocessor directives

|  |  |  |
| --- | --- | --- |
|  | Category | Preprocessor directives |
| 1 | Macro definitions | #define identifier replacement |
| 2 | Conditional inclusions | #ifdef, #ifndef, #if, #endif, #else and #elif |
| 3 | Line control | #line |
| 4 | Error directive | #error |
| 5 | Source file inclusion | #include |
| 6 | Pragma directive | #pragma |

# C/C++ data types

* Choose one option from each column to construct valid data type.
* If column provides an option of blank space, you can opt to choose that option as well.
* Options in the same column are mutually exclusive. For ex. signed and unsigned should not be chosen together.

## Use following table to form valid ‘int’ data types

|  |  |  |
| --- | --- | --- |
|  |  |  |
| unsigned | short | int |
| signed | long |

## Use following table to form valid ‘char’ data types

|  |  |
| --- | --- |
|  | char |
| unsigned |
| signed |

## Use following table to form valid ‘float’ data type

|  |  |
| --- | --- |
|  | float |
| long |

## Use following table to form valid ‘double’ data types

|  |  |
| --- | --- |
|  | double |
| long |

## Use following table to form valid ‘bool’ data type

|  |
| --- |
| bool |
|
|

## Use following table to form valid ‘wchar\_t’ data type

|  |
| --- |
| wchar\_t |
|
|

# Identifier naming rules

An identifier name must begin with a letter or an underscore and may be made up of any combinations of letters, underscores, or digits 0-9.

Whitespace characters (Space, Tab, and Newline) are not permitted within the name.

An identifier name may use uppercase letters or lowercase letters or both. Changing the case of even one character makes a different name.

The number of characters that can be used in an identifier name is compiler dependent.

C/C++ reserves certain names called keywords and should not be used for naming identifiers.

Refer <http://en.cppreference.com/w/cpp/keyword> link for complete list of C/C++ keywords.

https://www.tutorialspoint.com/How-many-keywords-are-there-in-Cplusplus

# Constants

## Integer constants

Decimal integer constants: 0 255 32767 32768 65535 2147483647

Octal integer constants: 012 037 0177 01000 077777 0100000

Hexadecimal integer constants: 0x2748 0x1f 0X1F 0xff 0xABC 0x100000 0x7FFFFFFF

## Floating point constants

Double precision (double) constants: 1.0 1. .1 0. .0

Single precision (float) constants: 1.0F 1.f .1F 0.f .0f

## Character constants

Character constants: ‘0’ ‘a’ ‘z’ ‘?’ ‘%’ ‘\a’ ‘\n’ ‘\\’ ‘\ooo’ ‘\xhh’

## String constants

String constants: “Madam, I’m Adam” “Programming in C++ is fun\n.”

Both

“Great things are not done by impulse, “

“but by a series of small things “

“brought together”

&

“Great things are not done by impulse \  
but by a series of small things brought together”

Is equivalent to -> “Great things are not done by impulse, but by a series of small things brought together”

Note: Adjacent string constants are concatenated at compile time.

# C/C++ operators

|  |  |
| --- | --- |
| Operator Description | Operator |
| Group 1 precedence, no associativity |  |
| Scope resolution | :: |
| Group 2 precedence, left to right associativity |  |
| Member selection (object or pointer) | . or –> |
| Array subscript | [ ] |
| Function call | ( ) |
| Postfix increment | ++ |
| Postfix decrement | -- |
| Type name | typeid( ) |
| Constant type conversion | const\_cast |
| Dynamic type conversion | dynamic\_cast |
| Reinterpreted type conversion | reinterpret\_cast |
| Static type conversion | static\_cast |
| Group 3 precedence, right to left associativity |  |
| Size of object or type | sizeof |
| Prefix increment | ++ |
| Prefix decrement | -- |
| One's complement | ~ |
| Logical not | ! |
| Unary negation | - |
| Unary plus | + |
| Address-of | & |
| Indirection | \* |
| Create object | new |
| Destroy object | delete |
| Cast | Cast: () |
| Group 4 precedence, left to right associativity |  |
| Pointer-to-member (objects or pointers) | .\* or –>\* |
| Group 5 precedence, left to right associativity |  |
| Multiplication | \* |
| Division | / |
| Modulus | % |
| Group 6 precedence, left to right associativity |  |
| Addition | + |
| Subtraction | – |
| Group 7 precedence, left to right associativity |  |
| Left shift | << |
| Right shift | >> |
| Group 8 precedence, left to right associativity |  |
| Less than | < |
| Greater than | > |
| Less than or equal to | <= |
| Greater than or equal to | >= |
| Group 9 precedence, left to right associativity |  |
| Equality | == |
| Inequality | != |
| Group 10 precedence left to right associativity |  |
| Bitwise AND | & |
| Group 11 precedence, left to right associativity |  |
| Bitwise exclusive OR | ^ |
| Group 12 precedence, left to right associativity |  |
| Bitwise inclusive OR | | |
| Group 13 precedence, left to right associativity |  |
| Logical AND | && |
| Group 14 precedence, left to right associativity |  |
| Logical OR | || |
| Group 15 precedence, right to left associativity |  |
| Conditional | ? : |
| Group 16 precedence, right to left associativity |  |
| Assignment | = |
| Multiplication assignment | \*= |
| Division assignment | /= |
| Modulus assignment | %= |
| Addition assignment | += |
| Subtraction assignment | –= |
| Left-shift assignment | <<= |
| Right-shift assignment | >>= |
| Bitwise AND assignment | &= |
| Bitwise inclusive OR assignment | |= |
| Bitwise exclusive OR assignment | ^= |
| Group 17 precedence, right to left associativity |  |
| throw expression | throw |
| Group 18 precedence, left to right associativity |  |
| Comma | , |

# List of standard header files

# Microsoft Visual Studio important programs, their meaning & their paths.

* Ensure that Windows Explorer shows file names with extensions.
* If you are using,
  + 64 bit Windows OS then traverse <drive>\Program Files (x86) sub tree [ <Program files> ]
  + 32 bit Windows OS then traverse <drive>\Program Files sub tree [ <Program files> ]
* If you are using
  + Microsoft Visual Studio 2008 then traverse <Program files>\Microsoft Visual Studio 9.0 [ <MSVS> ]
  + Microsoft Visual Studio 2010 then traverse <Program files>\Microsoft Visual Studio 10.0 [ <MSVS> ]
  + Microsoft Visual Studio 2012 then traverse <Program files>\Microsoft Visual Studio 11.0 [ <MSVS> ]
  + Microsoft Visual Studio 2013 then traverse <Program files>\Microsoft Visual Studio 12.0 [ <MSVS> ]

|  |  |  |
| --- | --- | --- |
| **Program name** | **Where is it located?** | **What does it do?** |
| devenv.exe | <MSVS>\Common7\IDE | Microsoft Visual Studio Integrated Development Environment (IDE)  Following features are provided by an IDE:   * Editor. * Invokes Microsoft C/C++ compiler and linker to build an application. * Execute an Application.Debug. |
| cl.exe | <MSVS>\VC\bin | Microsoft C/C++ Compiler |
| link.exe | <MSVS>\VC\bin | Microsoft Linker |
| lib.exe | <MSVS>\VC\bin | Microsoft Library Manager |

# Build process

## Stages in compilation

C/C++ Compiler

a.cpp

T

Source File

a.obj

0

Object File

Preprocessor

Compilation

## Building Executable from Single Source File

a.cpp

T

Source File

a.obj

0

Object File

a.exe

0

Executable

File

C/C++ Compiler

cl /c a.cpp

Linker

link a.obj

## Building Executable from Multiple Source Files

a.cpp

T

Source File

MS C/C++ Compiler

cl /c a.cpp

a.obj

0

Object File

MS Linker

link a.obj b.obj

a.exe

0

Executable

File

b.cpp

T

Source File

b.obj

0

Object File

MS C/C++ Compiler

cl /c b.cpp

## Build Executable from Source File and Static Library

a.cpp

T

Source File

C/C++ Compiler

cl /c a.cpp

a.obj

0

Object File

Linker

link a.obj b.obj

a.exe

0

Executable

File

b.cpp

T

Source File

C/C++ Compiler

cl /c b.cpp

b.obj

0

Object File

Library Manager

lib b.obj

b.lib

0

Library File

## Build Executable from Source File and DLL

Linker

link /DLL b.obj

b.exp

0

EXP File

b.dll

0

DLL

b.lib

0

Import Library File

b.cpp

T

Source File

C/C++ Compiler

cl /c b.cpp

b.obj

0

Object File

a.cpp

T

Source File

C/C++ Compiler

cl /c a.cpp

a.obj

0

Object File

Linker

link a.obj b.obj

a.exe

0

Executable

File