Searching

Two Basic Searches for arrays

- Sequential Search:
  - Can be used to locate an item in any array.
- Binary Search:
  - Requires an ordered list.

Sequential Search

- Used when the list is not ordered.
- Used for small lists or lists that are not searched often.
- Algorithm of Linear Search

Sequential Search

- Used when the list is not ordered.
- Used for small lists or lists that are not searched often.
- Algorithm of Linear Search

Algorithm of Linear Search

int linearSearch(int a[], int first,
int last, int key)

Searches a[first]..a[last] for key.

returns: index of the matching element if it finds key, otherwise -1.

a in array of (possibly unsorted) values.

first, last in lower and upper subscript bounds key in value to search for.

returns:

index of key, or -1 if key is not in the array.

1. Repeat for i=first to last
   a. if (key = a[i])
      display "key found at location I"

2. return -1; // failed to find key

Efficiency of linear search

Model	No. of Comparisions	Comparisions as a function of n
Best Case (fewer comparisions)	1(target is the first item)	1
Worst Case(most comparisions)	n(target is the last item)	n
Average Case(Avg no. of comparisons)	n/2(target is the middle item)	n/2

Disadvantage of sequential search

- Sequential search is very slow.
- Eg:

Binary Search

- Binary search is used when the list is sorted.
- It is used whenever the list starts to become large.(more than 15 elements).
- Working:

Algorithm of Binary Search

```
binarysearch(int a[],int
    value, int n)
//a[]:list of elements
//value:the value to be
    searched
//n:no. of elements in the list
Step 1:[initialize]
        first=0
        last =n-1
        flag=0
Step 2:
        repeat through step 4
        while(first<=last)
Step 3:
        mid=(first+last)/2
Step 4:
        if(value<a[mid])
                last=mid -1
        elseif(value>a[mid])
                first=mid+1
        elseif(value==a[mid])
                print "search
            successful"
            and location of
            the element mid
            flag=1
```

Algorithm of Binary Search

Step 5:

if(flag=0)

print "search is

unsuccessful)

Step 6:

Exit

Efficiency of binary search

Model	No. of Comparisions	Comparisions as a function of n
Best Case (fewer comparisions)	1(target is the middle item)	1
Average Case		Log_2 n
Worst Case(most comparisions)	n(target is the last item)	n

Indexed Sequential search

- Is another searching technique for a sorted list.
- In this an auxiliary “array index” is maintained in addition to the sorted list.
- Each element in the “array index” consists of a key value and a link to the record in the sorted list that corresponds to the key.
- Imp:
  - The elements in the “array index” as well as the elements in the original list must be sorted on the key.
- Example:
- Advantage:
  - Even if elements in the list are examined sequentially, the search time is sharply reduced as the search is performed on the smaller index rather than on the larger one.
  - Once the correct index position has been found in the original list, a second sequential search is performed on a smaller position of the original list itself.

Interpolation search

- Used for searching an ordered array.
- More efficient than the binary search, if the elements are sorted in a array.
- The key is expected to be at mid such that

  mid=first+(last –first) * [(key –A[first]) / A[last] -A[first])]
- Algorithm

Algorithm of Binary Search

interpolationsearch(int a[],int value, int n)

//a[]:list of elements

//value:the value to be searched

//n:no. of elements in the list

Step 1:[initialize]

first=0

last =n-1

flag=0

Step 2:

repeat through step 4

while(first<=last)

Step 3:

mid=first+(last –first) * [(key –A[first]) / A[last] -A[first])]

Step 4:

if(value<a[mid])

last=mid -1

elseif(value>a[mid])

first=mid+1

elseif(value==a[mid])

print “search successful”

and location of

the element mid

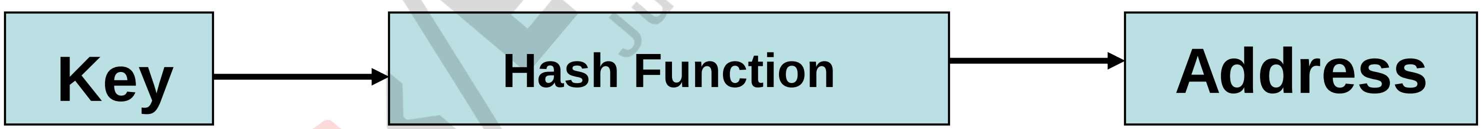
flag=1

Efficiency of interpolation search

Model	No. of Comparisions	Comparisions as a function of n
Best Case (fewer comparisions)	1(target is the middle item)	1
Average Case		Log_2 n
Worst Case(most comparisions)	n(target is the last item)	n

Hashed List Searches

- **Hashing or hash function** is a key-to-address transformation in which the keys map to the addresses in the list.
- Hashing is a key-to-address mapping process
- Diagram



Example:

Synonyms

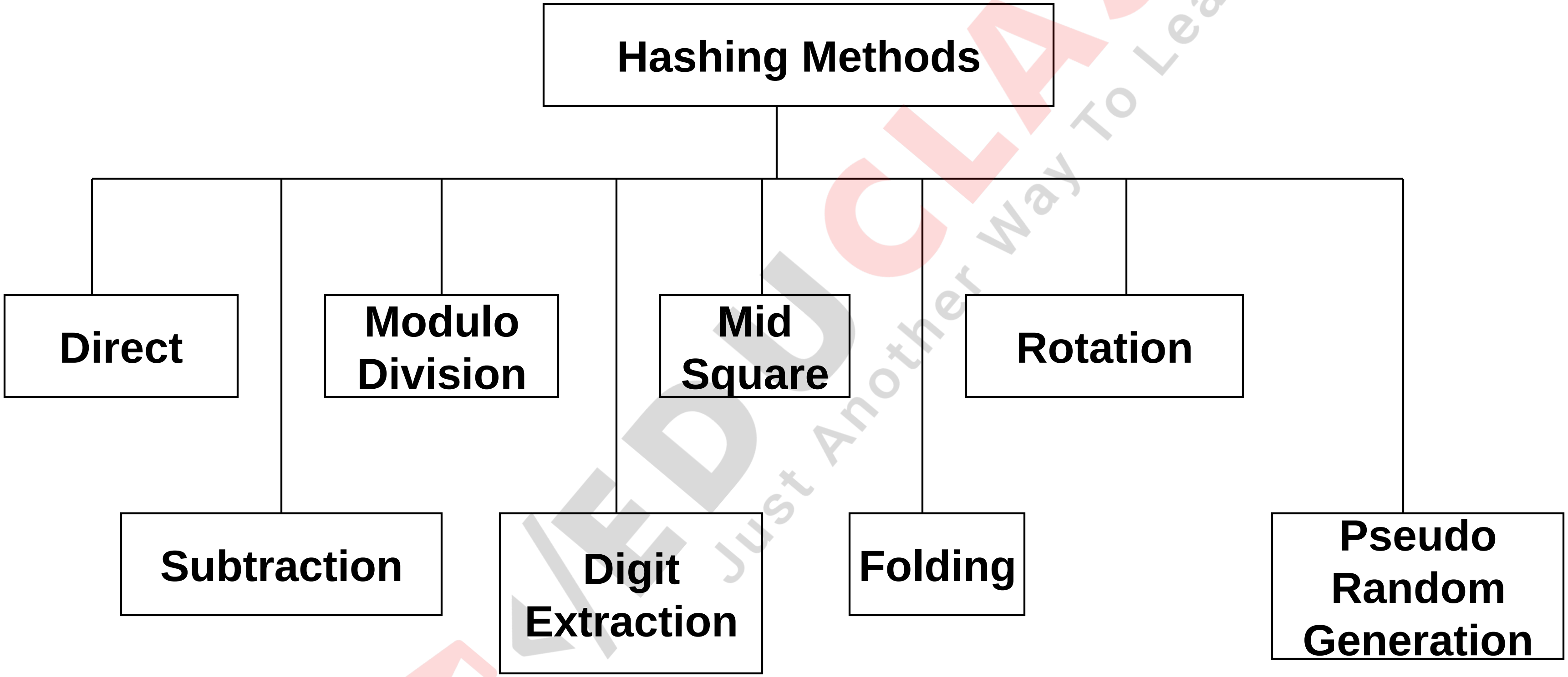
- The set of keys that hash to the same location is called a synonym.
- Example

Collision

- A **Collision** occurs when a hashing algorithm produces an address for a key and that address is already occupied.
- The address produced by the hashing algorithm is known as the **home address.**
- The memory that contains all of the home addresses is known as the **prime area.**

Hashing Methods

- Basic Hashing Techniques



Direct Hashing

- In direct hashing, **the key is the address without any algorithmic manipulation**.
- Therefore the data structure must contain an element for every possible key.
- **Advantage**
  - Applications of direct hashing are limited but can be powerful because there are **no synonyms and therefore no collisions.**

Cont…

Direct Hashing

- **Disadvantage**
  - Address space is as large as the key space
- Direct hashing is an ideal method but its application is very limited.It can be used only for small lists in which the keys map to a filled list
- Eg:

Subtraction Method

- In this method, the key is transformed to an address by subtracting a fixed number from it.
- It is simple and guarantees that there will be no collisions.
- Limitations:
- limited.It can be used only for small lists in which the keys map to a filled list

Similarity between Direct Hashing and Subtraction Method

- The **direct hashing** and **substraction methods** both guarantee search with **no collisions.**They are **one-to-one hashing methods.i.e. only one key hashes to each address**

Modulo-Division Method

- Also known as **division remainder**, this method **divides the key by the array size.and uses the remainder for the address.**

address = key MOD listsize

when address range from **0 to listsize-1**

Or

address = (key MOD listsize) +1

when address range from **1 to listsize**

Modulo-Division Method

- This method works with **any list size**.However a **list size that is a prime number produces fewer collisions** than other list sizes.Therefore **make the array size a prime number.**
- Example

Digit Extraction Method

- In this method, selected digits are extracted from the key and used as the address.
- Example:
  - using six-digit employee number to hash to a three-digit address (000–999), we could select the first, third and fourth digits (from the left) and use them as the address.

MidSquare Method

- In midsquare hashing , the key is squared and the address is selected from the middle of the squared numbers.
- **Advantage:**
  - Entire key is used to calculate the address, reducing chances of collisions

MidSquare Method

- **Disadvantage:**
  - The size of the key.Eg. If a key is 6 digits, the product will be 12 digits which is beyond the max integer size of many computers.
- **Variation of Mid Square Method**
  - **Select a portion of the key such as the first 3 digits and then use the midsquare method.**

Folding Methods

- Two folding methods are used
  - Fold shift
  - Fold boundary
- **Fold Shift**
  - **In fold shift,** the key value is divided into parts such that the

    size of the parts = size of the required address
  - **The left and right parts are shifted and added with the middle part.**
  - **If sum>size of the address , discard the leading digits.**

- Fold Boundary:
  - The left and right numbers **are folded on a fixed boundary between them and centre number.The two outside values are thus reversed.**
  - **examples**

Rotation Method

- Rotation hashing is **not used by itself** but is incorporated in combination with other hashing methods.
- **Most useful when key are assigned serially.**
- **A simple hashing algorithm** tends to create **synonyms when hashing keys are identical except for the last character.**
- **Rotating the last character to the front of the key minimizes this effect.**

Rotation Method

- Modula division method do not work well with rotation method.
- Rotation is used only in combination with folding and pseudorandom hashing

Pseudorandom hashing

- In this the key is used as the seed in a pseudorandom number generator.
- The resulting random number is then scaled into the possible range using modulo-division method.
- A common random number generator is

  $$y=ax + c$$

  x = key

  a and c = factors that should be prime numbers since prime numbers minimize collisions
- Example

Multiplicative Method

- The hashing technique uses the following formula:
- h(key) =floor(m * frac(c* key))

  where floor =integer part of real number

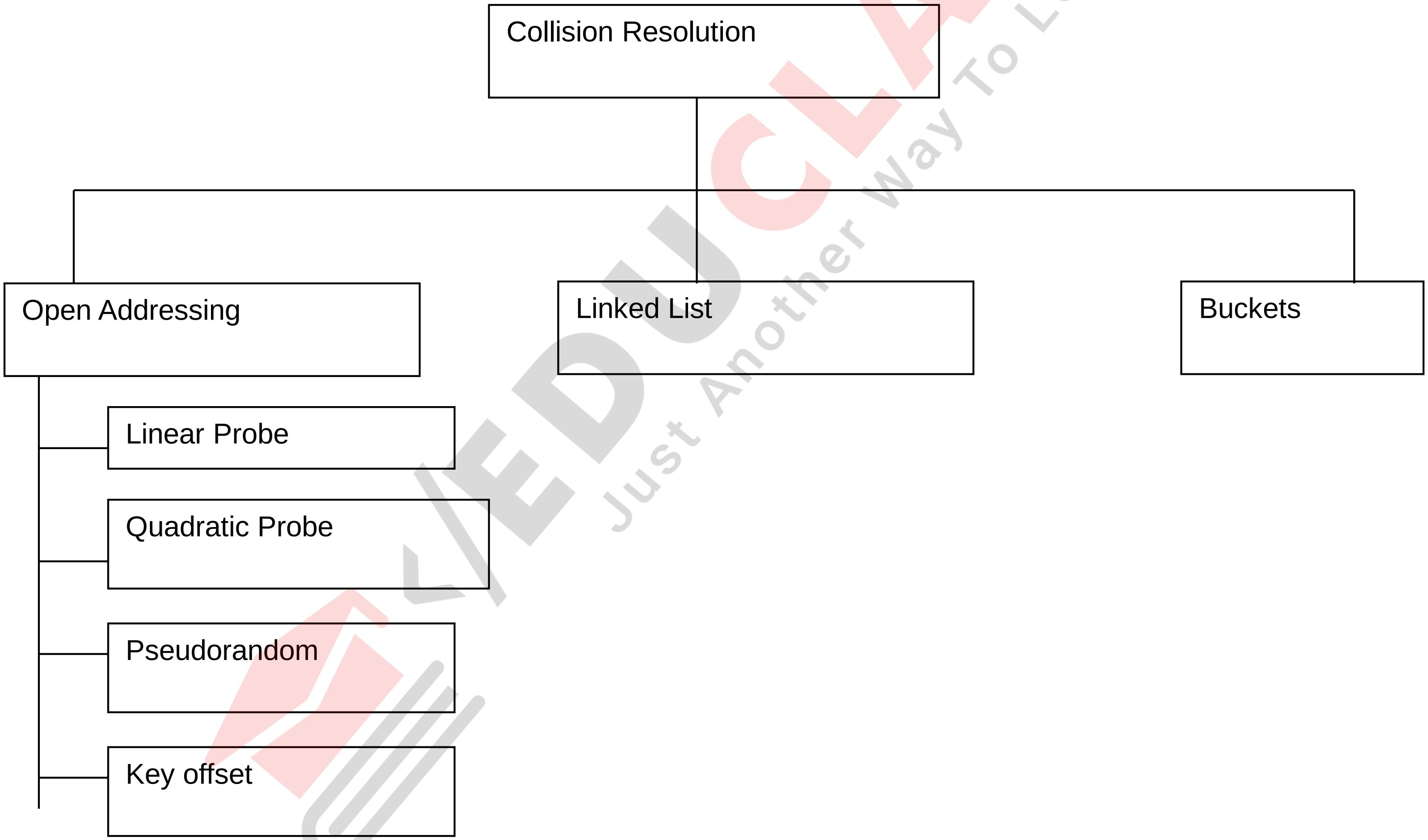
  frac(x)= fractional part

  c= 0.618 , yields good theoritical properties
- Disadvantage:
  - Slower than modulo division method

IMP NOTE

- **All hash functions** except **direct hashing and subtraction hashing** are in such a way **that "many keys hash to one address"**

Collision Resolution



Concepts for collision resolution methods:

- **Load factor:**
  - The load factor(**α-alpha**)of a hashed list is :

    $$\frac{\text{Number of elements in the list}}{\text{Number of physical elements allocated for the list}} * 100$$

  - Fullness of a file is measured by its **load factor**
  - When the address space of a relative file gets full, the probability of collision arises dramatically.
  - **A load factor 70% or 80% gives reasonable performance.**
  - **Example. If a file contains n records, the address space should have room for storing 1.25n records(80%)**

Clustering

- **Clustering:**
  - As the data are added to the list ,some hashing algorithms tend to cause data to group within the list.
  - This tendency of data to build up unevenly across a hashed list is known as **clustering.**
  - **Clustering is usually created by collisions.**

Types of clustering

- Two types of clusters exist:
  - **Primary clustering:**
    - Occur **when data cluster around a home address.**
      - **The collision resolution is based on home address**
      - **Easy to identify.**
      - **Primary cluster slows down the operations**
      - **Example**

Types of clustering

- **Secondary clustering:**
  - **The collision resolution is not based on the home address.**
  - **The collision resolution algorithm spreads the collisions across the entire list.**
  - **Not easy to identify**
  - **The time to locate a requested element of data becomes faster.**
  - **Example**

Open Addressing:

– Resolves collisions in the prime area i.e. the area that contains all of the home addresses.
– When a collision occurs, the prime area addresses are searched for an unoccupied element where the data can be placed.

Linear Probe

- In a linear probe, when data cannot be stored in the home address, the collision is resolved by adding 1 to the current address.
- Example
- As an alternative to a simple linear probe, we can add 1, subtract 2, add 3 subtract 4 and so forth until an element is located.

Linear Probe

- We must ensure that the next collision resolution address lies within the boundaries of the list. Eg: if a key hashes to the last location in the list, adding 1 must produce the address of the first element. Similarly if a key hashes to the first location in the list, subtracting 1 must produce the address of the last element.
- Advantages:
  - Simple to implement.
  - Data tend to remain near their home address.
  - Linear probes tend to produce primary clustering.
  - Linear probes tend to make the search algorithm more complex.

Quadratic probe:

- **In the quadratic probe,**
  - **the increment = the collision probe number squared**
  - **The new address = collision location + increment**
  - **Disadvantage:**
    - **Time required to square the probe number. Therefore instead of multiplication factor, we can use an increment factor that increases by 2 with each probe.**
    - **It is not possible to generate a new address for every element in the list.**

Double Hashing

The pseudorandom collision resolution and key offset

pseudorandom collision resolution and key offset

- In each method, rather than use an arithmetic probe function, the address is rehashed
- Both methods prevent primary clustering.

Pseudorandom collision resolution

- **Uses pseudorandom number to resolve the collision.**
- **In this, rather than use the key as a factor in the random-number calculation, we use the collision address.**
- **Advantage:**
  - **Pseudorandom collision resolution have simple solution**
  - **Produces only one collision resolution path through the list.**

Key offset:

- It is a double hashing method that produces different collision paths for different keys.
- The pseudorandom number generator produces a new address as a function of the previous address, key offset produces a new address as a function of the previous address and the key.
- Example
- Therefore each key resolves its collision at a different address

Linked List Collision resolution

- Major disadvantage of open addressing is that each collision resolution increases the probability of future collisions. This disadvantage is eliminated by Linked list collision resolution.
- Example
- Linked list collision resolution uses separate area to store collisions and chains all synonyms together in a linked list.

Linked List Collision resolution

- Uses two storage areas :prime area and overflow area :
  - Each element in the prime area contains an additional field- a link head pointer to a linked list of overflow data in the overflow area.
  - When collision occurs , one element is stored in the prime area and chained to its corresponding linked list in the overflow area.
  - The linked list data can be stored in any order, but a LIFO sequence is the most common as it the fastest.

Bucket Hashing

- The keys are hashed to buckets. The buckets are nodes that accommodate multiple data occurrences.
- Because a bucket can hold multiple data, collisions are postponed until the bucket is full.
- Problems in bucket hashing:
  - It uses more space because many of the buckets are empty or partially empty at any given time.
  - It does not completely resolve the collision problem.
- Example

Application of Hashing

- Hash tables are good in situations where you have enormous amounts of data from which you would like to quickly search and retrieve information.
- A few typical hash table implementations would be in the following situations:
  - For driver's license record's. With a hash table, you could quickly get information about the driver (ie. name, address, age) given the licence number.
  - For compiler symbol tables. The compiler uses a symbol table to keep track of the user-defined symbols in a C++ program. This allows the compiler to quickly look up attributes associated with symbols (for example, variable names)
  - For internet search engines.
  - For telephone book databases. You could make use of a hash table implementation to quickly look up John Smith's telephone number.
  - For electronic library catalogs. Hash Table implementations allow for a fast find among the millions of materials stored in the library.
  - For implementing passwords for systems with multiple users. Hash Tables allow for a fast retrieval of the password which corresponds to a given username.

Hash Table - Windows Internet Explorer provided by SIESCOMS

http://www.cs.uregina.ca/Links/class-info/210/Hash/

2. Application: Looking up Passwords

The following section outlines an algorithm for authenticating a user's password. Later, in the lab exercise, you will be given the skeleton code and asked to add lines to make it work.

One possible use for a hash table is to store computer user login usernames and passwords.

There are two major steps to this program:

1. The program will load username/password sets from the file `password.dat` and insert them into the hash table until the end of file is reached on `password.dat`. The password.dat file will look something like this with one username/password set per line:

```
jack    broken.crown
jill    tumblin'down
mary    contrary
bopeep  sheep!lost
```

2. The program will then present a login prompt, read one username, present a password prompt, and after looking up the username's password in the hash table, will print either "Authentication successful" or "Authentication failure". The output might look something like this:

```
Login: mary
Password: contrary
Authentication successful

Login: jim
Password: contrary
Authentication failure

Login: bopeep
Password: sheeplost
Authentication failure
```

Step 2 will be repeated until the end of the input data (EOF) is reached on the console input stream (cin). The EOF character on the PC's is the CTRL Z character.

- Symbol table:
  - The symbol table records information about each *symbol name* in a program.
  - Many compilers set up a table for the various variables in the program and fill in the information about the symbol later during semantic analysis when more information about the variable is known

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL