CSE231: Data Structures

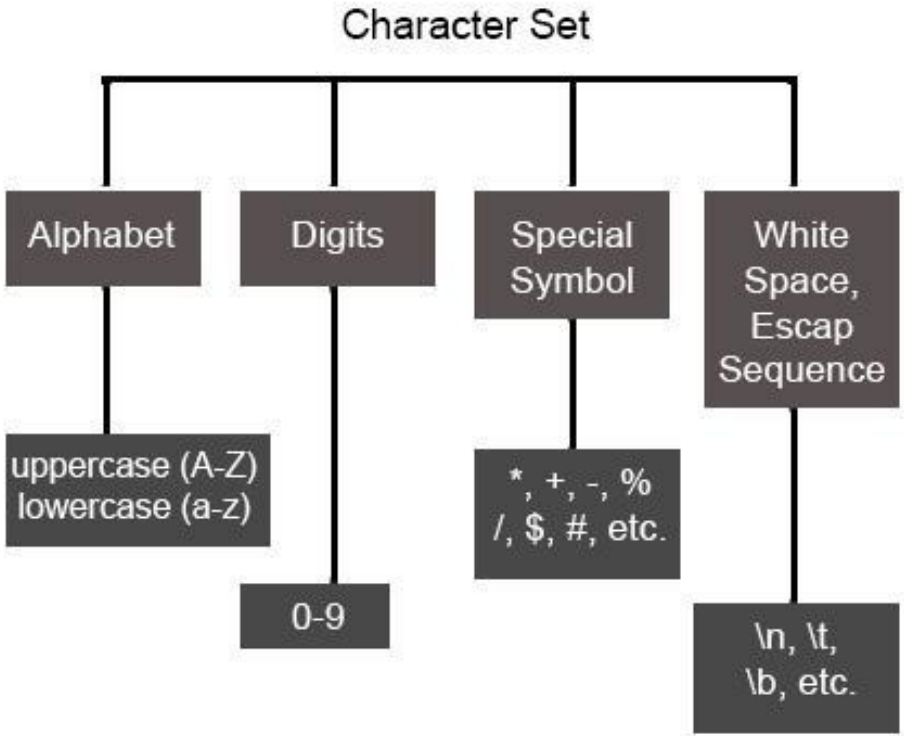
Lecture 05

by

Mehedi Hasan Bijoy

CHARACTER SET

It refers to a set of all the valid characters that we can use in the source program for forming words, expressions, and numbers.



CHARACTER SET

Type of Character	Description	Characters
Lowercase Alphabets	a to z	a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z
Uppercase Alphabets	A to Z	A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z
Digits	0 to 9	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special Characters	–	` ~ @ ! $ # ^ * % & ( ) [ ] { } < > + = _ – [/ \ ; : ‘ “ , . ?
White Spaces	–	Blank Spaces, Carriage Return, Tab, New Line

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

STRING

- A finite sequence S of zero or more characters is called a string.
  - A sequence or linear array of characters.
- The number of characters in a string is called its length.
- The string with zero characters is called the empty/null string.

Syntax

```
string str_name = "This is a C++ string";
```

Or

```
char str_array[7] = {'S', 'c', 'a', 'l', 'e', 'r', '\0'};
```

STRING CONCATENATION

Let S1 and S2 be strings:

S1 = ‘THE END’ S2 = ‘TO BE OR NOT TO BE’

- The string consisting of the characters of S1 followed by the characters of S2 is called the concatenation of S1 and S2 which is denoted as S1 // S2.

  S1 // S2 = ‘THE ENDTO BE OR NOT TO BE’

- More Examples:

'THE' // 'END' = 'THEEND' but 'THE' // ' □ ' // ' END = 'THE END'

STRING CONCATENATION

S = X // Y // Z (Consider X, Y, and Z are strings)

- If X is an empty string, they Y is called an initial substring of S.
- If Z is an empty string, they Y is called an terminal substring of S.

'BE OR NOT' is a substring of 'TO BE OR NOT TO BE'
'THE' is an initial substring of 'THE END'

STORING STRINGS

Strings are stored in three types of structures:

- Fixed-length structures
- Variable-length structures with fixed maximums
- Linked structures

STORING STRINGS: FIXED-LENGTH

- In fixed-length storage each line of print is viewed as a record, where all records have the same length.

Advantages:

- The ease of accessing data from any given record
- The ease of updating data in any given record (as long as the length of the new data does not exceed the record length)

STORING STRINGS: FIXED-LENGTH

- In fixed-length storage each line of print is viewed as a record, where all records have the same length.

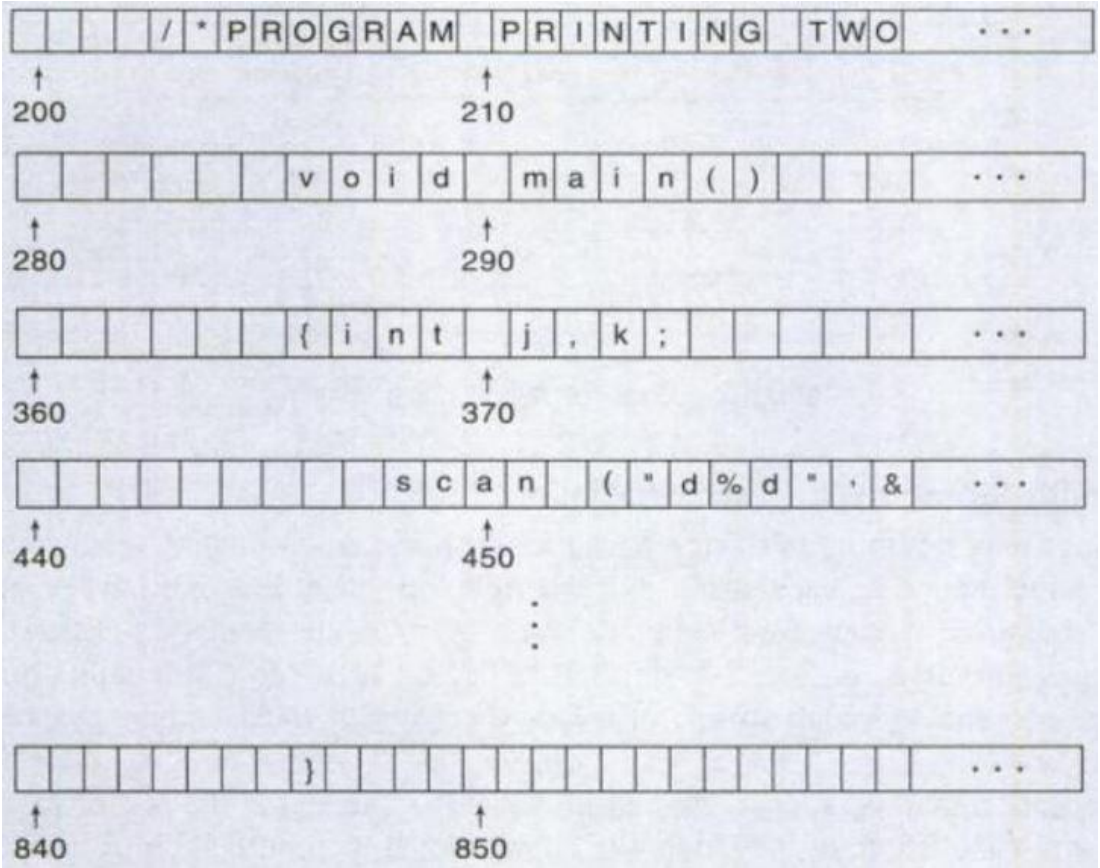
Advantages:

- The ease of accessing data from any given record
- The ease of updating data in any given record (as long as the length of the new data does not exceed the record length)

Disadvantages:

- Time is wasted reading an entire record if most of the storage consists of inessential blank spaces.
- Certain records may require more space than available.

STORING STRINGS: FIXED-LENGTH



STORING STRINGS: VARIABLE-LENGTH with Fixed Maximum

- The storage of variable-length strings in memory cells with fixed lengths can be done in two ways:
  - (Method- 1) One can use a marker, such as two dollar sign ($$), to signal the end of string.
  - (Method- 2) One can list the length of the string – as an additional item in the pointer array.

STORING STRINGS: VARIABLE-LENGTH with Fixed Maximum

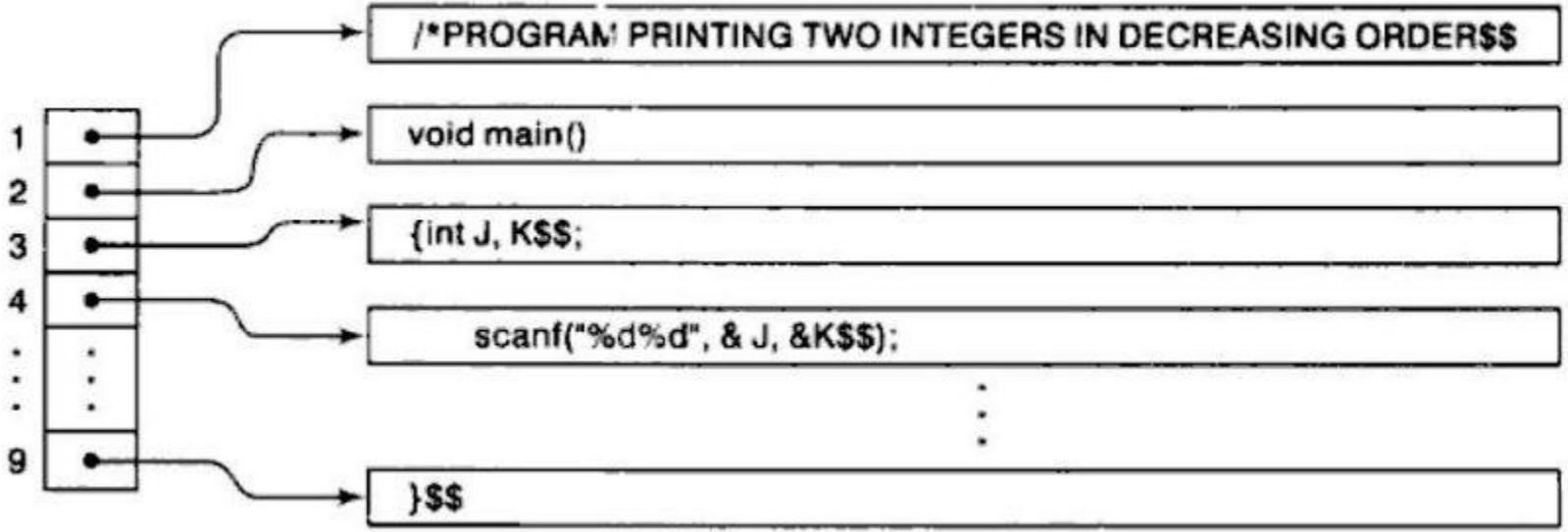


Fig: Method- 1

STORING STRINGS: VARIABLE-LENGTH with Fixed Maximum

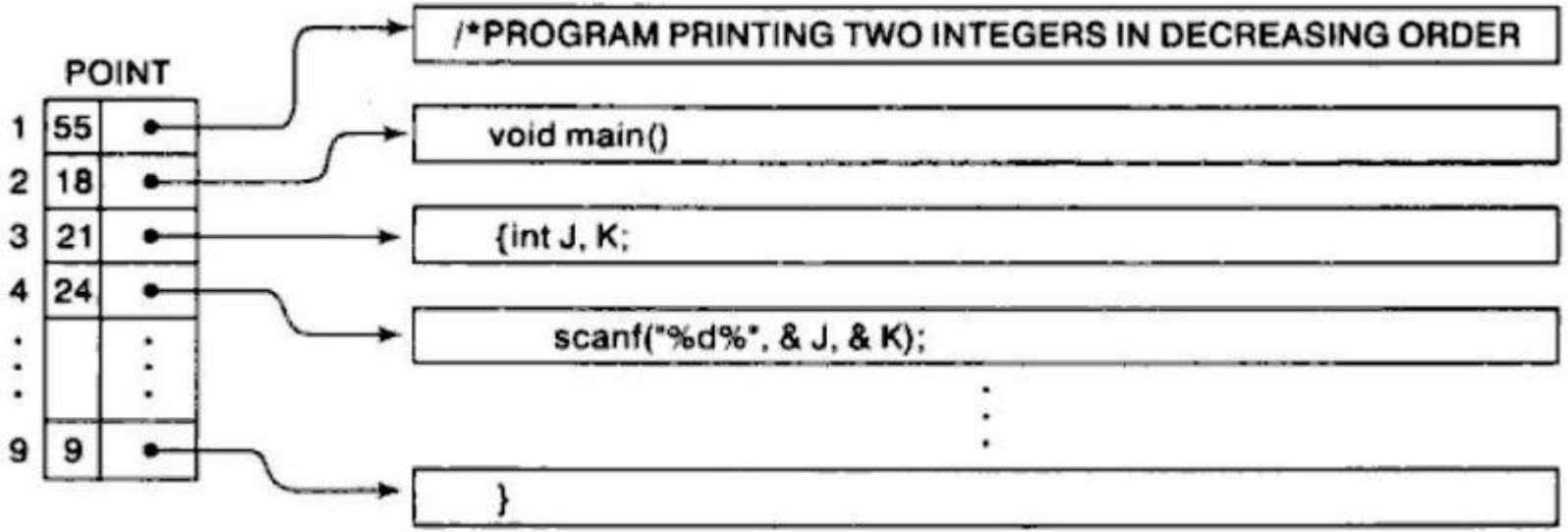
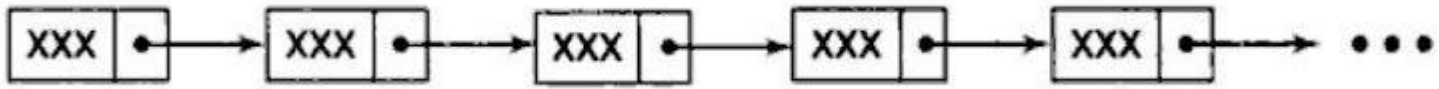


Fig: Method- 2

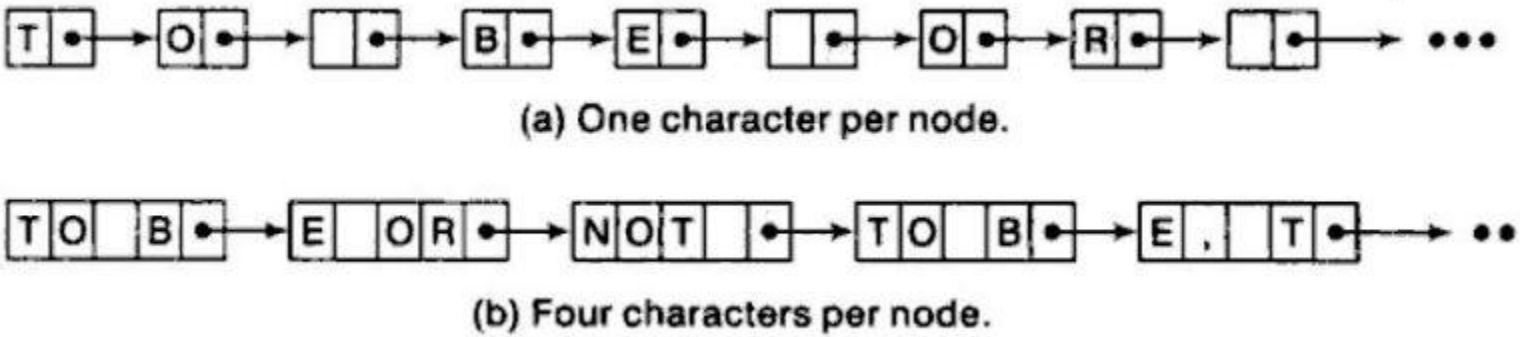
STORING STRINGS: LINKED

- For most extensive word processing applications, strings are stored by means of linked lists.
- By a linked list, we mean a linearly ordered sequence of memory cells, called nodes, where each node contain an item and a link which points to the next node.



STORING STRINGS: LINKED

- Strings can be stored in linked lists as follows:



(a) One character per node.

(b) Four characters per node.

STRING AS ADT

We can implement our own string data type

- The use of a data type in string processing applications should not depend on how it is implemented, or whether it is build-in or user defined.
- We only need to know what operations are allowed on the string.

STRING AS ADT

The following set of operations we might want to do on strings:

GETCHAR(*str*, *n*)	Returns the n^{th} character in the string
PUTCHAR(*str*, *n*, *c*)	Sets the n^{th} character in the string to *c*
LENGTH(*str*)	Returns the number of characters in the string
POS(*str*1, *str*2)	Returns the position of the first occurrence of *str*2 found in *str*1, or 0 if no match
CONCAT(*str*1, *str*2)	Returns a new string consisting of characters in *str*1 followed by characters in *str*2
SUBSTRING(*str*1, *i*, *m*)	Returns a substring of length *m* starting at position *i* in string *str*
DELETE(*str*, *i*, *m*)	Deletes *m* characters from *str* starting at position *i*
INSERT(*str*1, *str*2, *i*)	Changes *str*1 into a new string with *str*2 inserted in position *i*
COMPARE(*str*1, *str*2)	Returns an integer indicating whether *str*1 > *str*2

STRING OPERATIONS

Substring:

- A group of consecutive elements in a string is called substring.

  String: ‘TO BE OR NOT TO BE’

  Substring: TO, BE, OR, BE OR, TO BE

STRING OPERATIONS

Substring:

Accessing a substring from a given string requires three pieces of information:

- Name of the string
- The position of the first character of the substring in the given string
- The length of the substring

SUBSTRING(string_name, initial_position, length)

```
SUBSTRING('TO BE OR NOT TO BE', 4, 7) = 'BE OR N'
SUBSTRING('THE END' , 4, 4) = '□END'
```

STRING OPERATIONS

Indexing:

- Indexing, also called pattern matching, refers to finding the position where a string pattern P first appears in a given string text T.
- We call this operation INDEX and write as follows:

  INDEX(text, pattern)

- If the pattern P does not appear in the text T, then INDEX is assigned the value 0.

STRING OPERATIONS

Indexing:

Suppose T contains the text

'HIS FATHER IS THE PROFESSOR'

Then,

INDEX(T, 'THE'), INDEX(T, 'THEN') and INDEX(T, '□THE□')

have the values 7, 0 and 14, respectively.

STRING OPERATIONS

Concatenation:

- Let S1 and S2 be strings. The concatenation of S1 and S2, which is denoted as S1 // S2, is the string consisting of the characters of S1 followed by the characters of S2.

Suppose S_1 = 'MARK' and S_2 = 'TWAIN'. Then:

$S_1//S_2$ = 'MARKTWAIN' but S_1// '□'//S_2 = 'MARK TWAIN'

STRING OPERATIONS

Length:

- The number of characters in a string is called its length.

Suppose S = 'COMPUTER'. Then:

LENGTH (S) = 8

Similarly,

LENGTH ('MARC TWAIN') = 10

Thank You!