STRING

Algorithm Problem Solving – Samsung Vietnam R&D Center

Compose by phuong.ndp

Agenda

- 2D Array
- Sequential Search
- Binary Search
- Selection Sort
- Problem Solving

ASCII / UNICODE

Characters Before ASCII/Unicode

Fundamentally, computers just deal with numbers. They store letters and other characters by assigning a number for each one. Before Unicode was invented, there were hundreds of different systems, called character encodings, for assigning these numbers.

These early character encodings were limited and could not contain enough characters to cover all the world's languages. Even for a single language like English no single encoding was adequate for all the letters, punctuation, and technical symbols in common use.

Early character encodings also conflicted with one another. That is, two encodings could use the same number for two different characters, or use different numbers for the same character. Any given computer (especially servers) would need to support many different encodings.

However, when data is passed through different computers or between different encodings, that data runs the risk of corruption.

Brief History of ASCII code

The American Standard Code for Information Interchange, or ASCII code, was created in 1963 by the "American Standards Association" Committee or "ASA", the agency changed its name in 1969 by "American National Standards Institute" or "ANSI" as it is known since.

This code arises from reorder and expand the set of symbols and characters already used in telegraphy at that time by the Bell company.

At first only included capital letters and numbers , but in 1967 was added the lowercase letters and some control characters, forming what is known as US-ASCII (Standard ASCII), ie the characters 0 through 127.

So with this set of only 128 characters was published in 1967 as standard, containing all you need to write in English language.

Standard ASCII can represent 128 characters. It uses 7 bits to represent each character since the first bit of the byte is always 0. For instance, a capital "T" is represented by 84, or 01010100 in binary. A lowercase "t" is represented by 116 or 01110100 in binary.

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

Source: www.LookupTables.com

Compose by phuong.ndp

Brief History of ASCII code

In 1981, IBM developed an extension of 8-bit ASCII code, called "code page 437", in this version were replaced some obsolete control characters for graphic characters. Also 128 characters were added , with new symbols, signs, graphics and latin letters, all punctuation signs and characters needed to write texts in other languages, such as Spanish.

In this way was added the ASCII characters ranging from 128 to 255. (Extended ASCII)

The 128 (2^7) characters supported by Standard ASCII are enough to represent all standard English letters, numbers, and punctuation symbols.

However, it is not sufficient to represent all special characters and characters from other languages.

Extended ASCII helps solve this problem by adding an extra 128 values, for a total of 256 (2^8) characters. The additional binary values start with a 1 instead of a 0.

For example, in extended ASCII, the character "é" is represented by 233, or 11101001 in binary.

Extended ASCII is programmable; characters are based on the language of your operating system or program you are using. Foreign letters are also placed in this section.

128	Ç	144	É	160	á	176	░	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

Source: www.LookupTables.com

Compose by phuong.ndp

What is Unicode?

In computer systems, characters are transformed and stored as numbers (sequences of bits) that can be handled by the processor. A code page is an encoding scheme that maps a specific sequence of bits to its character representation. The pre-Unicode world was populated with hundreds of different encoding schemes that assigned a number to each letter or character. Many such schemes included code pages that contained only 256 characters - each character requiring 8 bits of storage.

While this was relatively compact, it was insufficient to hold ideographic character sets containing thousands of characters such as **Vietnamese** and Japanese, and also did not allow the character sets of many languages to co-exist with each other.

Unicode is an attempt to include all the different schemes into one universal text-encoding standard.

A Brief Introduction to Unicode for Everybody

https://blog.daftcode.pl/a-brief-introduction-to-unicode-for-everybody-bc26f5761548

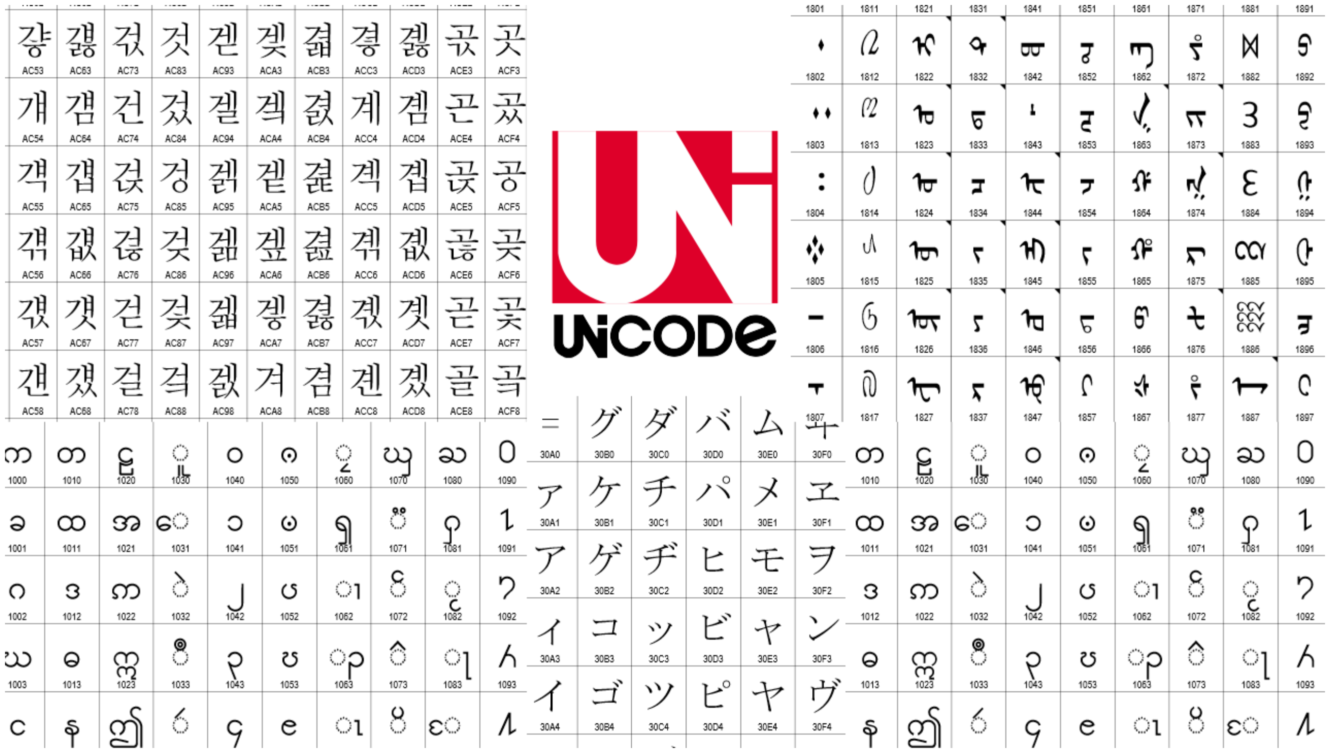
Compose by phuong.ndp

The importance of Unicode

Unicode represents a mechanism to support more regionally popular encoding systems

From a translation/localization point of view, Unicode is an important step towards standardization, at least from a tools and file format standpoint.

- Unicode enables a single software product or a single website to be designed for multiple platforms, languages and countries (no need for re-engineering) which can lead to a significant reduction in cost over the use of legacy character sets.
- Unicode data can be used through many different systems without data corruption.
- Unicode represents a single encoding scheme for all languages and characters.
- Unicode is a common point in the conversion between other character encoding schemes. Since it is a superset of all of the other common character encoding systems, you can convert from one encoding scheme to Unicode, and then from Unicode to the other encoding scheme.
- Unicode is the preferred encoding scheme used by XML-based tools and applications.



Bottom line: Unicode is a worldwide character-encoding standard, published by the Unicode Consortium. Computers store numbers that represent a character; Unicode provides a unique number for every character.

STRING IN C/C++

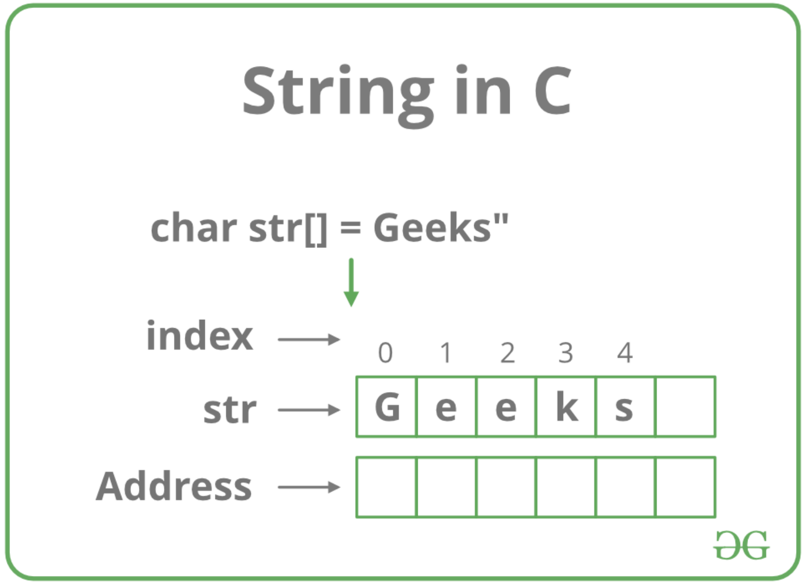
String in C

Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character '\0'.

Declaration of strings: Declaring a string is as simple as declaring a one dimensional array. Below is the basic syntax for declaring a string.

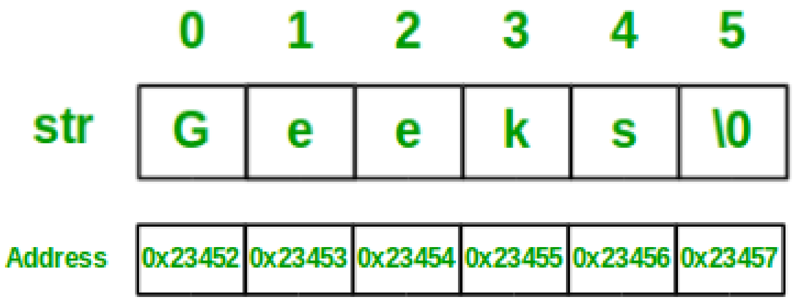
char str_name[size];

Please keep in mind that there is an extra terminating character which is the **Null** character ('**\0**') used to indicate termination of string which differs strings from normal character arrays.



Initializing a String: A string can be initialized in different ways. We will explain this with the help of an example. Below is an example to declare a string with name as str and initialize it with "GeeksforGeeks".

```
char str[] = "GeeksforGeeks";
char str[50] = "GeeksforGeeks";
char str[] = {'G','e','e','k','s','f','o','r','G','e','e','k','s','\0'};
char str[14] = {'G','e','e','k','s','f','o','r','G','e','e','k','s','\0'};
```



Reading String & Passing to function

Below is a sample program to read a string from user:

```c
// C program to read strings

#include<stdio.h>

int main()
{
    // declaring string
    char str[50];

    // reading string
    scanf("%s",str);

    // print string
    printf("%s",str);

    return 0;
}
```

Passing strings to function: As strings are character arrays, so we can pass strings to function in a same way we pass an array to a function. Below is a sample program to do this:

```c
// C program to illustrate how to
// pass string to functions
#include<stdio.h>

void printStr(char str[])
{
    printf("String is : %s",str);
}

int main()
{
    // declare and initialize string
    char str[] = "GeeksforGeeks";

    // print string by passing string
    // to a different function
    printStr(str);

    return 0;
}
```

Output:

```
String is : GeeksforGeeks
```

Read a Word

Notice that, in the second example only "Programming" is displayed instead of "Programming is fun".

This is because the extraction operator **>>** works as **scanf()** in C and considers a space " " has a terminating character.

Example 1: C++ String to read a word

C++ program to display a string entered by user.

```
#include <iostream>
using namespace std;

int main()
{
    char str[100];

    cout << "Enter a string: ";
    cin >> str;
    cout << "You entered: " << str << endl;

    cout << "\nEnter another string: ";
    cin >> str;
    cout << "You entered: "<<str<<endl;

    return 0;
}
```

Output

```
Enter a string: C++
You entered: C++

Enter another string: Programming is fun.
You entered: Programming
```

Read a Line of Text

To read the text containing blank space, **cin.get** function can be used. This function takes two arguments.

First argument is the **name of the string** (address of first element of string) and second argument is the **maximum size of the array**.

In the above program, str is the name of the string and 100 is the maximum size of the array.

Example 2: C++ String to read a line of text

C++ program to read and display an entire line entered by user.

```
#include <iostream>
using namespace std;

int main()
{
    char str[100];
    cout << "Enter a string: ";
    cin.get(str, 100);

    cout << "You entered: " << str << endl;
    return 0;
}
```

Output

```
Enter a string: Programming is fun.
You entered: Programming is fun.
```

Functions for manipulating C strings

No	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.
4	**strcmp(s1, s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
5	**strchr(s1, ch);** Returns a pointer to the first occurrence of character ch in string s1.
6	**strstr(s1, s2);** Returns a pointer to the first occurrence of string s2 in string s1.

```
strcpy( str3, str1) : Hello
strcat( str1, str2): HelloWorld
strlen(str1) : 10
```

```
#include <iostream>
#include <cstring>
using namespace std;
int main () {
char str1[10] = "Hello";
char str2[10] = "World";
char str3[10];
int len ;

// copy str1 into str3
strcpy( str3, str1);
cout << "strcpy( str3, str1) : " << str3 << endl;

// concatenates str1 and str2
strcat( str1, str2); cout << "strcat( str1, str2): "
<< str1 << endl;

// total lenghth of str1 after concatenation
len = strlen(str1);
cout << "strlen(str1) : " << len << endl;

return 0;
}
```

String Class in C++

C++ provides following two types of string representations –

- The C-style character string.
- The string class type introduced with Standard C++.

C++ has in its definition a way to represent sequence of characters as an object of class. This class is called std:: string. String class stores the characters as a sequence of bytes with a functionality of allowing access to single byte character.

```
#include <iostream>
#include <string>

using namespace std;

int main () {

   string str1 = "Hello";
   string str2 = "World";
   string str3;
   int  len ;

   // copy str1 into str3
   str3 = str1;
   cout << "str3 : " << str3 << endl;

   // concatenates str1 and str2
   str3 = str1 + str2;
   cout << "str1 + str2 : " << str3 << endl;

   // total length of str3 after concatenation
   len = str3.size();
   cout << "str3.size() :  " << len << endl;

   return 0;
}
```

```
str3 : Hello
str1 + str2 : HelloWorld
str3.size() :  10
```

Character Array vs std:: string

A character array is simply **an array of characters** can terminated by a null character. A string is a **class which defines objects** that be represented as stream of characters.

Size of the character array has to **allocated statically**, more memory cannot be allocated at run time if required. Unused allocated **memory is wasted** in case of character array. In case of strings, memory is **allocated dynamically**. More memory can be allocated at run time on demand. As no memory is preallocated**, no memory is wasted**.

Implementation of **character array is faster** than std:: string. **Strings are slower** when compared to implementation than character array.

Character array **do not offer** much **inbuilt functions** to manipulate strings. String class defines **a number of functionalities** which allow manifold operations on strings.

Input String Data Type

In this program, a **string** str is declared. Then the string is asked from the user.

Instead of using **cin>>** or **cin.get()** function, you can get the entered line of text using **getline()**.

getline() function takes the input stream as the first parameter which is **cin** and str as the location of the line to be stored.

Example 3: C++ string using string data type

```
#include <iostream>
using namespace std;

int main()
{
    // Declaring a string object
    string str;
    cout << "Enter a string: ";
    getline(cin, str);

    cout << "You entered: " << str << endl;
    return 0;
}
```

Output

```
Enter a string: Programming is fun.
You entered: Programming is fun.
```

STRING IN JAVA

String in Java

In Java, strings are special. For example, to create the string objects you **need not to use 'new'** keyword. Where as to create other type of objects you **have to use 'new'** keyword.

JVM divides the allocated memory to a Java program into two parts. one is Stack and another one is heap. Stack is used for execution purpose and heap is used for storage purpose.

In that heap memory, JVM allocates some memory specially meant for string literals. This part of the heap memory is called String Constant Pool.

Whenever you create a string object **using string literal**, that object is stored in the string constant pool and whenever you create a string object using **new** keyword, such object is stored in the heap memory.

String Constant Pool.

```java
String s1 = "abc";

String s2 = "xyz";

String s3 = "123";

String s4 = "A";
```

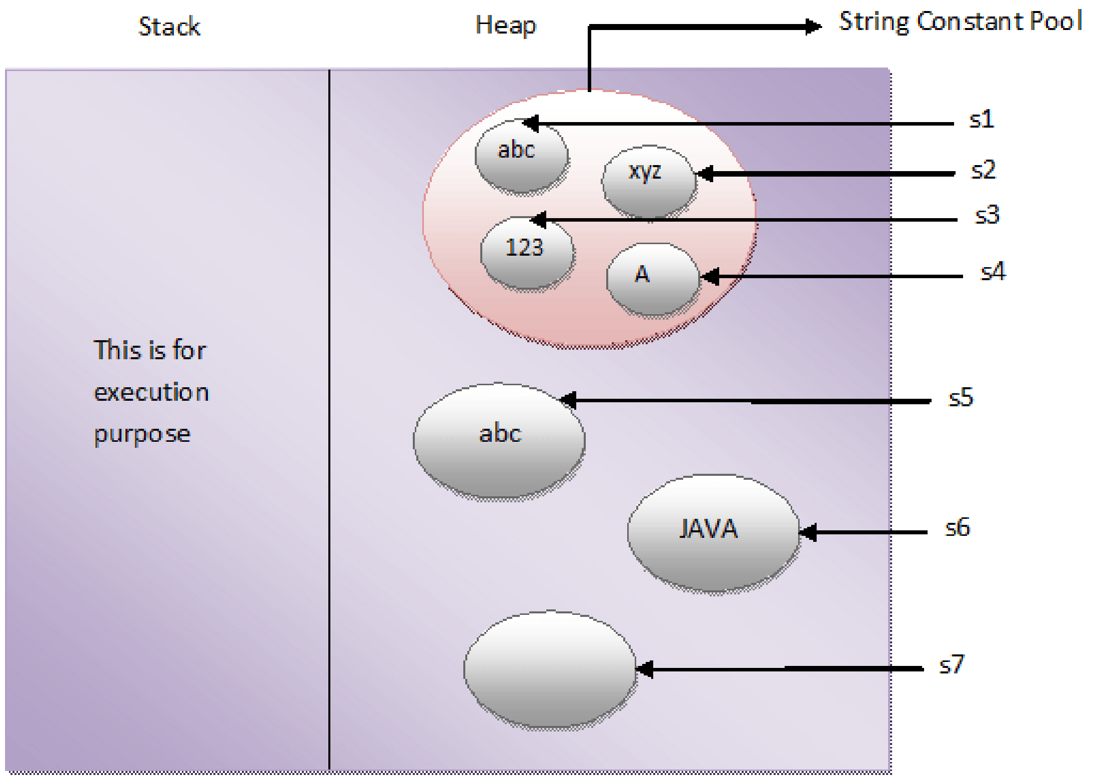
Heap memory.

```java
String s5 = new String("abc");

char[] c = {'J', 'A', 'V', 'A'};

String s6 = new String(c);

String s7 = new String(new StringBuffer());
```



String in Java

Pool space is allocated to an object depending upon it's content. There will be **no two objects** in the pool having the **same content**.

This is what happens when you create string objects using string literal,

"When you create a string object using string literal, JVM first checks the content of to be created object. If there exist an object in the pool with the same content, then it returns the reference of that object. It doesn't create new object. If the content is different from the existing objects then only it creates new object."

When you create string objects using new keyword, a new object is created **whether the content is same or not**.

This can be proved by using "==" operator. As "==" operator returns true if two objects have **same physical address** in the memory otherwise it will return false.

In simple words, there can not be two string objects with same content in the string constant pool. But, there can be two string objects with the same content in the heap memory.

```
public class StringExamples
{
    public static void main(String[] args)
    {
        //Creating string objects using literals

        String s1 = "abc";

        String s2 = "abc";

        System.out.println(s1 == s2);        //Output : true

        //Creating string objects using new operator

        String s3 = new String("abc");

        String s4 = new String("abc");

        System.out.println(s3 == s4);        //Output : false
    }
}
```

Thank you!

Algorithm Problem Solving – Samsung Vietnam R&D Center

Compose by phuong.ndp

Source

https://theasciicode.com.ar/

https://techterms.com/definition/ascii

https://www.computerhope.com/jargon/a/ascii.htm

https://www.interproinc.com/blog/unicode-101-introduction-unicode-standard

https://unicode.org/standard/WhatIsUnicode.html

https://www.geeksforgeeks.org/strings-in-c-2/

https://www.tutorialspoint.com/cplusplus/cpp_strings.htm

https://www.programiz.com/cpp-programming/strings

https://javaconceptoftheday.com/how-the-strings-are-stored-in-the-memory/#:~:text=This%20part%20of%20the%20heap,stored%20in%20the%20heap%20memory.

https://thuytrangcoding.wordpress.com/2018/02/11/string-match-kmp/

https://stackjava.com/algorithm/thuat-toan-tim-kiem-knuth-morris-pratt.html