



Strings

# Representing Characters: Unicode. ASCII

---

- Printable characters in a string are stored as bits in a computer, just like int and double variables
- The bit patterns are standardized:
  - ASCII (American Standard Code for Information Interchange) is 7 bits long, specifying  $2^7 = 128$  codes:
    - 26 uppercase letters A through Z
    - 26 lowercase letters a through z
    - 10 digits
    - 32 typographical symbols such as +, -, ', \...
    - 34 control characters such as space, newline
    - 32 others for controlling printers and other devices.
- Unicode, which has replaced ASCII in most cases, is 21 bits superset of ASCII; the first 128 codes match. The extra bits allow many more characters ( $2^{21} > 2 \times 10^6$ ), required for worldwide languages

# ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]	48	30	110000	60	0	96	60	1100000	140	`
1	1	1	1	[START OF HEADING]	49	31	110001	61	1	97	61	1100001	141	a
2	2	10	2	[START OF TEXT]	50	32	110010	62	2	98	62	1100010	142	b
3	3	11	3	[END OF TEXT]	51	33	110011	63	3	99	63	1100011	143	c
4	4	100	4	[END OF TRANSMISSION]	52	34	110100	64	4	100	64	1100100	144	d
5	5	101	5	[ENQUIRY]	53	35	110101	65	5	101	65	1100101	145	e
6	6	110	6	[ACKNOWLEDGE]	54	36	110110	66	6	102	66	1100110	146	f
7	7	111	7	[BELL]	55	37	110111	67	7	103	67	1100111	147	g
8	8	1000	10	[BACKSPACE]	56	38	111000	70	8	104	68	1101000	150	h
9	9	1001	11	[HORIZONTAL TAB]	57	39	111001	71	9	105	69	1101001	151	i
10	A	1010	12	[LINE FEED]	58	3A	111010	72	:	106	6A	1101010	152	j
11	B	1011	13	[VERTICAL TAB]	59	3B	111011	73	;	107	6B	1101011	153	k
12	C	1100	14	[FORM FEED]	60	3C	111100	74	<	108	6C	1101100	154	l
13	D	1101	15	[CARRIAGE RETURN]	61	3D	111101	75	=	109	6D	1101101	155	m
14	E	1110	16	[SHIFT OUT]	62	3E	111110	76	>	110	6E	1101110	156	n
15	F	1111	17	[SHIFT IN]	63	3F	111111	77	?	111	6F	1101111	157	o
16	10	10000	20	[DATA LINK ESCAPE]	64	40	1000000	100	@	112	70	1110000	160	p
17	11	10001	21	[DEVICE CONTROL 1]	65	41	1000001	101	A	113	71	1110001	161	q
18	12	10010	22	[DEVICE CONTROL 2]	66	42	1000010	102	B	114	72	1110010	162	r
19	13	10011	23	[DEVICE CONTROL 3]	67	43	1000011	103	C	115	73	1110011	163	s
20	14	10100	24	[DEVICE CONTROL 4]	68	44	1000100	104	D	116	74	1110100	164	t
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]	69	45	1000101	105	E	117	75	1110101	165	u
22	16	10110	26	[SYNCHRONOUS IDLE]	70	46	1000110	106	F	118	76	1110110	166	v
23	17	10111	27	[ENG OF TRANS. BLOCK]	71	47	1000111	107	G	119	77	1110111	167	w
24	18	11000	30	[CANCEL]	72	48	1001000	110	H	120	78	1111000	170	x
25	19	11001	31	[END OF MEDIUM]	73	49	1001001	111	I	121	79	1111001	171	y
26	1A	11010	32	[SUBSTITUTE]	74	4A	1001010	112	J	122	7A	1111010	172	z
27	1B	11011	33	[ESCAPE]	75	4B	1001011	113	K	123	7B	1111011	173	{
28	1C	11100	34	[FILE SEPARATOR]	76	4C	1001100	114	L	124	7C	1111100	174	
29	1D	11101	35	[GROUP SEPARATOR]	77	4D	1001101	115	M	125	7D	1111101	175	}
30	1E	11110	36	[RECORD SEPARATOR]	78	4E	1001110	116	N	126	7E	1111110	176	~
31	1F	11111	37	[UNIT SEPARATOR]	79	4F	1001111	117	O	127	7F	1111111	177	[DEL]
32	20	100000	40	[SPACE]	80	50	1010000	120	P					
33	21	100001	41	!	81	51	1010001	121	Q					
34	22	100010	42	"	82	52	1010010	122	R					
35	23	100011	43	#	83	53	1010011	123	S					
36	24	100100	44	\$	84	54	1010100	124	T					
37	25	100101	45	%	85	55	1010101	125	U					
38	26	100110	46	&	86	56	1010110	126	V					
39	27	100111	47	'	87	57	1010111	127	W					
40	28	101000	50	{	88	58	1011000	130	X					
41	29	101001	51	}	89	59	1011001	131	Y					
42	2A	101010	52	*	90	5A	1011010	132	Z					
43	2B	101011	53	+	91	5B	1011011	133	[					
44	2C	101100	54	,	92	5C	1011100	134	\					
45	2D	101101	55	-	93	5D	1011101	135	]					
46	2E	101110	56	.	94	5E	1011110	136	^					
47	2F	101111	57	/	95	5F	1011111	137	_					

# Strings

# Strings

---

- Strings are sequences of characters:

```
"Hello world"
```

- Include the string header, so you can create variables to hold strings:

```
#include <iostream>
#include <string>
using namespace std;
...
string name = "Harry";
           // literal string "Harry" stored
```

# String Initializations

---

- String variables are automatically initialized to the empty string if you don't initialize them:

```
string response;  
    // literal string "" stored  
    // it is not garbage
```

- "" is called the empty or null string.

# Concatenation of Strings

---

- Use the `+` operator to *concatenate* strings;  
that is, put them together to yield a longer string.

```
string f_name = "Harry";  
string l_name = "Potter";  
string n_name = f_name + l_name; //need a space!  
cout << name << endl;  
name = f_name + " " + l_name; //got a space  
cout << n_name << endl;
```

The output will be:

```
HarryPotter  
Harry Potter
```

# Common Error – Concatenation of literal strings

---

```
string greeting = "Hello, " + " World!";  
                // will not compile
```

Literal strings cannot be concatenated. And it's pointless anyway, just do:

```
string greeting = "Hello World!";
```



# String Input

---

- You can read a string from the console:

```
cout << "Please enter your name: ";  
string name;  
cin >> name;
```

- When a string is read with the `>>` operator, only one word is placed into the `string` variable.
- For example, suppose the user types  
Harry Potter  
as the response to the prompt.
- Only the string "Harry" is placed into the variable name.

# String Input

---

You can use another input string to read the second word:

```
cout << "Please enter your name: ";  
string f_name, l_name;  
cin >> f_name >> l_name;
```

```
//f_name gets Harry, l_name gets Potter
```

# String Input

---

`getline()` function allows us to accept a full string input

```
cout << "Please enter your name: ";  
string name;  
getline(cin, name);
```

```
//name gets Harry Potter
```

# String Functions

---

- The `length` *member function* yields the number of characters in a string.
- Unlike the `sqrt` or `pow` function, the `length` function is *invoked* with the *dot notation*:

```
string name = "Harry";  
int n = name.length();
```

# String Data Representation & Character Positions

---

H	e	l	l	o	,		W	o	r	l	d	!
0	1	2	3	4	5	6	7	8	9	10	11	12

- In most computer languages, the starting position 0 means “start at the beginning.”
- The first position in a string is labeled 0, the second 1, and so on. And don’t forget to count the space character after the comma—but the quotation marks are **not** stored.
- The position number of the last character is always one less than the length of the **string**.

# substr Function

---

- Once you have a string, you can extract substrings by using the **substr** member function.
- `s.substr(start, length)`  
returns a string that is made from the characters in the string `s`, starting at character `start`, and containing `length` characters. (`start` and `length` are integers)
  - NOTE: the first character has an index of 0, not 1.

```
string greeting = "Hello, World!";  
string sub = greeting.substr(0, 2);  
    // sub contains "He"
```

# Another Example of the `substr` Function

---

```
string greeting = "Hello, World!";  
string w = greeting.substr(7, 5);  
    // w contains "World" (not the !)
```

- "World" is 5 characters long but...
- Why is 7 the position of the "W" in "World"?
- Why is the "W" not @ 8?
- *Because the first character has an index of 0, not 1.*

# String Character Positions

---

H	e	l	l	o	,		W	o	r	l	d	!
0	1	2	3	4	5	6	7	8	9	10	11	12

```
string greeting = "Hello, World!";  
string w = greeting.substr(7);  
// w contains "World!"
```

- If you do not specify how many characters should go into the substring, the call to the **substr()** function will return a substring that starts at the specified index, and goes until the end of the string



## String Operations Examples

Statement	Result	Comment
string str = "C"; str = str + "++";	str is set to "C++"	When applied to strings, + denotes concatenation.
string str = "C" + "++";	Error	Error: You cannot concatenate two string literals.
cout << "Enter name: "; cin >> name; (User input: Harry Morgan)	name contains "Harry"	The >> operator places the next word into the string variable.
cout << "Enter name: "; cin >> name >> last_name; (User input: Harry Morgan)	name contains "Harry", last_name contains "Morgan"	Use multiple >> operators to read more than one word.
string greeting = "H & S"; int n = greeting.length();	n is set to 5	Each space counts as one character.
string str = "Sally"; string str2 = str.substr(1, 3);	str2 is set to "all"	Extracts the substring of length 3 starting at position 1. (The initial position is 0.)
string str = "Sally"; string str2 = str.substr(1);	str2 is set to "ally"	If you omit the length, all characters from the position until the end are included.
string a = str.substr(0, 1);	a is set to the initial letter in str	Extracts the substring of length 1 starting at position 0.
string b = str.substr(str.length() - 1);	b is set to the last letter in str	The last letter has position str.length() - 1. We need not specify the length.

# String Functions, Complete Program Example

---

ch02/initials.cpp

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    cout << "Enter your first name: ";
    string first;
    cin >> first;
    cout << "Enter your significant other's first name: ";
    string second;
    cin >> second;
    string initials = first.substr(0, 1) + "&" + second.substr(0, 1);
    cout << initials << endl;
    return 0;
}
```