Programmētāju skola
3. līmeņa grupa

simbolu rindas

Simbolu kodēšana

Katrs simbols datora atmiņā glabājas veselā skaitļa veidā un uzzināt, ka šīs skaitlis apzīmē simbolu, nevis kaut ko citu, praktiski neiespējami.

A → 65

* → 42

! → 33

A + 1 → B

* + ! → K

* * 2 → T

ASCII

ASCII (**American Standard Code for Information Interchange**) – amerikāņu informācijas apmaiņas standartkods – simbolu kodēšanas tabula, kurā dažiem drukājamām un nedrukājamām simboliem ir piešķirti skaitliskie kodi. Tabula tika izstrādāta un standartizēta ASV 1963.gadā un saturēja 128 simbolu kodus (7 bitu kodēšana).

NUL 0	SOH 1	STX 2	ETX 3	EOT 4	ENQ 5	ACK 6	BEL 7	BS 8	HT 9	LF 10	VT 11	FF 12	CR 13	SO 14	SI 15
DLE 16	DC1 17	DC2 18	DC3 19	DC4 20	NAK 21	SYN 22	ETB 23	CAN 24	EM 25	SUB 26	ESC 27	FS 28	GS 29	RS 30	US 31
SP 32	! 33	" 34	# 35	$ 36	% 37	& 38	' 39	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95
` 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	| 124	} 125	~ 126	DEL 127

Kodu lappuses

Mūsdienu kodēšanas tabulas tiek paplašinātas līdz 256 simbolu kodus (8 bitu kodēšana), kur pirmās puses simboli (0–127) ir nemainīgie ASCII simboli, bet otras puses simboli (128–255) ir atkarīgie no izmantojamās kodēšanas lappuses (code page).

cp1251 – Windows Cyrillic

Ђ 128	Ѓ 129	‚ 130	ѓ 131	„ 132	… 133	† 134	‡ 135	€ 136	‰ 137	Љ 138	‹ 139	Њ 140	Ќ 141	Ћ 142	Џ 143
ђ 144	‘ 145	’ 146	“ 147	” 148	• 149	– 150	— 151	152	™ 153	љ 154	› 155	њ 156	ќ 157	ћ 158	џ 159
160	Ў 161	ў 162	Ј 163	¤ 164	Ґ 165	¦ 166	§ 167	Ё 168	© 169	Є 170	« 171	¬ 172	173	® 174	Ї 175
° 176	± 177	І 178	і 179	ґ 180	µ 181	¶ 182	· 183	ё 184	№ 185	є 186	» 187	ј 188	Ѕ 189	ѕ 190	ї 191
А 192	Б 193	В 194	Г 195	Д 196	Е 197	Ж 198	З 199	И 200	Й 201	К 202	Л 203	М 204	Н 205	О 206	П 207
Р 208	С 209	Т 210	У 211	Ф 212	Х 213	Ц 214	Ч 215	Ш 216	Щ 217	Ъ 218	Ы 219	Ь 220	Э 221	Ю 222	Я 223
а 224	б 225	в 226	г 227	д 228	е 229	ж 230	з 231	и 232	й 233	к 234	л 235	м 236	н 237	о 238	п 239
р 240	с 241	т 242	у 243	ф 244	х 245	ц 246	ч 247	ш 248	щ 249	ъ 250	ы 251	ь 252	э 253	ю 254	я 255

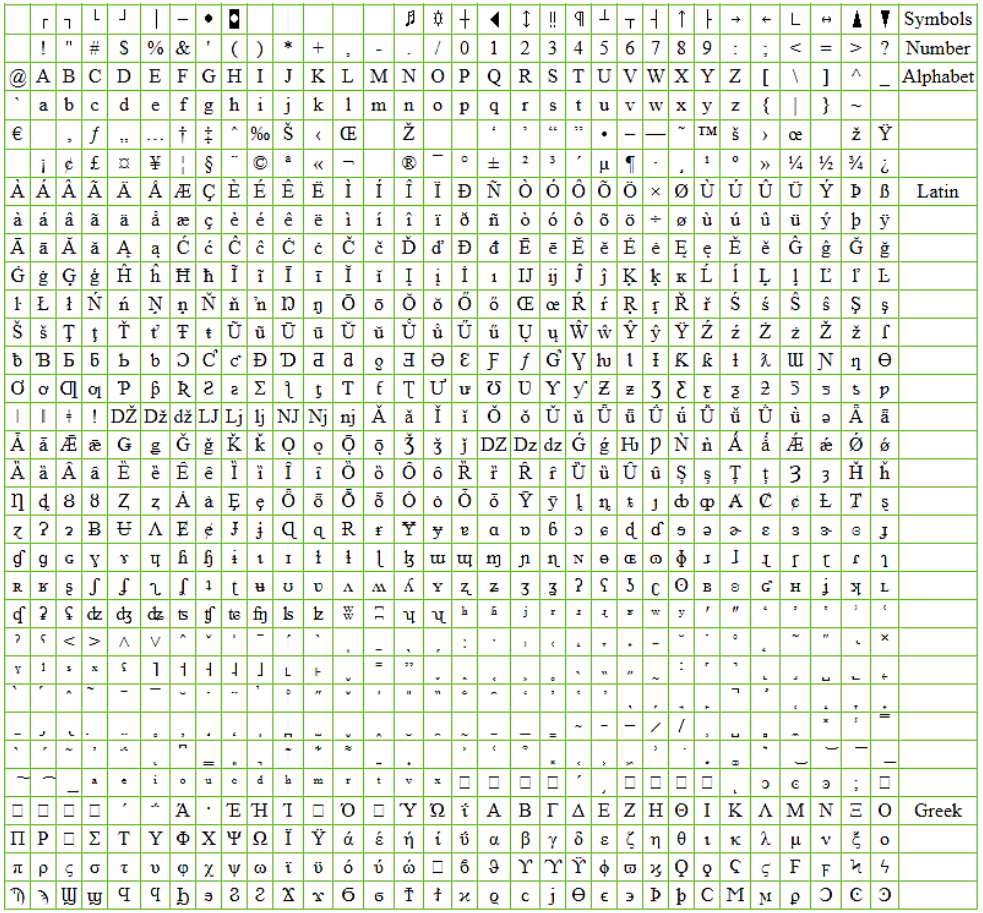
cp1257 - Windows Baltic

€ 128	129	‚ 130	131	„ 132	… 133	† 134	‡ 135	136	‰ 137	138	‹ 139	140	¨ 141	ˇ 142	¸ 143
144	‘ 145	’ 146	“ 147	” 148	• 149	– 150	— 151	152	™ 153	143	› 155	156	¯ 157	˛ 158	159
NBSP 160	161	¢ 162	£ 163	¤ 164	165	¦ 166	§ 167	Ø 168	© 169	Ŗ 170	« 171	¬ 172	SHY 173	® 174	Æ 175
° 176	± 177	² 178	³ 179	´ 180	µ 181	¶ 182	· 183	ø 184	¹ 185	ŗ 186	» 187	¼ 188	½ 189	¾ 190	æ 191
Ą 192	Į 193	Ā 194	Ć 195	Ä 196	Å 197	Ę 198	Ē 199	Č 200	É 201	Ź 202	Ė 203	Ģ 204	Ķ 205	Ī 206	Ļ 207
Š 208	Ń 209	Ņ 210	Ó 211	Ō 212	Õ 213	Ö 214	× 215	Ų 216	Ł 217	Ś 218	Ū 219	Ü 220	Ż 221	Ž 222	ß 223
ą 224	į 225	ā 226	ć 227	ä 228	å 229	ę 230	ē 231	č 232	é 233	ź 234	ė 235	ģ 236	ķ 237	ī 238	ļ 239
š 240	ń 241	ņ 242	ó 243	ō 244	õ 245	ö 246	÷ 247	ų 248	ł 249	ś 250	ū 251	ü 252	ż 253	ž 254	˙ 255

	1251	1257	1252	878
207	= П	= Ļ	= Ï	= ο = …
248	= Ш	= ų	= ø	= Ь = …
254	= ю	= ž	= þ	= Ч = …

UNICODE

UNICODE – simbolu kodēšanas standarts, kas paredzēts visu pasaules valodu rakstzīmju kodēšanai. Izmantojot 16 bitu kodus, var tikt kodētas 65536 rakstzīmes. Unikoda pirmie 128 kodi ir identiski ASCII kodiem.



П = U+041F = 1055

Ļ = U+013B = 319

Ï = U+00CF = 207

о = U+043E = 1086

Ш = U+0448 = 1096

ц = U+0173 = 371

ø = U+00F8 = 248

ь = U+042C = 1068

ю = U+044E = 1102

ž = U+017E = 382

þ = U+00FE = 254

Ч = U+0427 = 1063

Simbolu konstantes un mainīgie

Simbolu literāli

drukājamā forma

nedrukājamā forma

atsoļa sekvences (escape sequences)

```
'T' = 84   'K' = 75   '*' = 42   '!' = 33
'\n' = 10  '\r' = 13  '\b' = 8   '\t' = 9
'\0' = 0   '\a' = 7   '\0x41' = 'A' = 65
'\\' = 92  '\'' = 39  '\"' = 34
```

Simbolu mainīgie

```
char ch;
char ch1 = 'A', ch2 = 65;
char ch = -100;
char ch = 'A'; ch++;
char ch = 'Z'; ch += 'a' - 'A';
char ch = '8'; int d = ch - '0';
char ch = '*' + '!';  // 42 + 33 = 75 = 'K'
char ch = '*' * 2;  // 42 * 2 = 84 = 'T'
```

Standarta simbolu funkcijas

C nodrošina standarta funkciju kopumu darbam ar simboliem. Lai tos pielietotu, programmai jāpievieno bibliotēkas virsraksts **<cctype>**

`isalpha(chr)`	pārbauda, vai simbols `chr` ir alfabēta burts
`isdigit(chr)`	pārbauda, vai simbols `chr` ir cipars
`ispunct(chr)`	pārbauda, vai simbols `chr` ir pieturzīme
`isspace(chr)`	pārbauda, vai simbols `chr` ir atstarpes rakstzīme
`islower(chr)`	pārbauda, vai simbols `chr` ir mazais burts
`isupper(chr)`	pārbauda, vai simbols `chr` ir lielais burts
`tolower(chr)`	pārvērš simbolu `chr` par mazo burtu
`toupper(chr)`	pārvērš simbolu `chr` par lielo burtu

```
char chr = '5';
if(isdigit(chr)) chr++; //6
```

```
char chr = 'a';
if(isalpha(chr)) chr = toupper(chr); //A
```

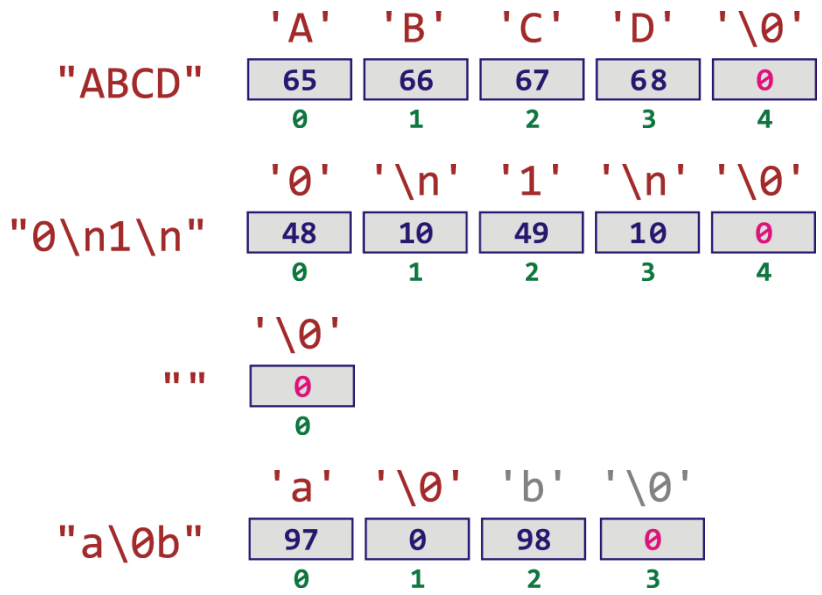
Simbolu rindas konstantes

Literāli

```
"Hello, world!\n"    "5"    ""    "d:\\Student\\SchoolC\\"
```

Glabāšana atmiņā

Katrs rindas simbols atmiņā tiek glabāts vienbaita ASCII koda veidā. Lai atzīmētu rindas beigas, pēc pēdējā simbola pievieno **vēl vienu simbolu ar nulles kodu**. Tādu rindas glabāšanas formātu sauc par null-terminated string jeb ASCIIZ string.



Simbolu rindas mainīgie

C++ atbalsta divas pieejas simbolu rindas glabāšanai mainīgajos:
C stils (`char[]` jeb `char*`) un **C++ stils** (`std::string`).

C style: `#include <cstring>`

Simbolu rindas glabāšanai **C valodā** izmanto simbolu **masīvu:**

```
char str[8] = "ABCD";
char str[9];
char str[] = "ABCD";
```

C++ style: `#include <string>`

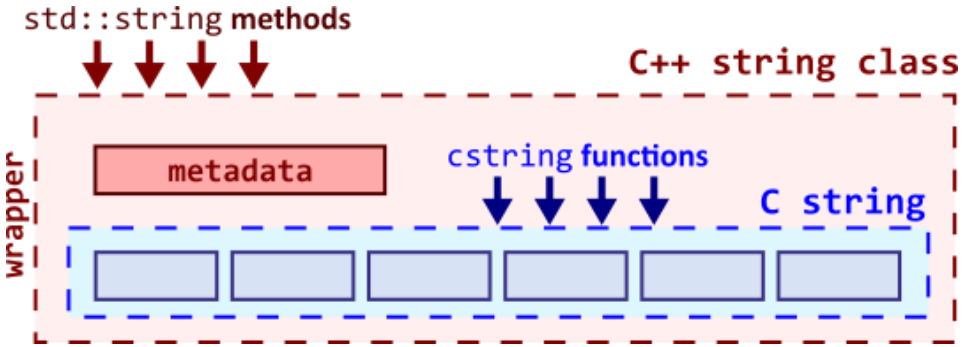
C++ valodā rindas glabāšanai var izmantot **std::string** klases objektu:

```
std::string str;
std::string str = "ABCD";
```

Jebkurā gadījumā atmiņā rinda glabājas **ASCIIZ masīvā**

```
char str[8] = "ABCD"; ≈ std::string str = "ABCD";
 'A'  'B'  'C'  'D'  '\0'
```

65	66	67	68	0	0	0	0
0	1	2	3	4	5	6	7



Rindas simbolus piekļuve

Simbolu rinda – tā ir **simbolu masīvs**, kurā var piekļūt katru simbolu atsevišķi, noradot tās indeksu kvadrātiekavās.

C style: `str[id]`

```
char str[8] = "ABCD";
str[0] → 'A'
str[4] → 0
str[1] = 'b'; → str = "AbCD"
str[2] = 0; → str = "Ab"
```

C++ style: `str[id]`

```
std::string str = "ABCD";
str[0] → 'A'
str[4] → 0
str[1] = 'b'; → str = "AbCD"
str[2] = 0; → str = "Ab\0D"
```

Lai apstrādātu katru rindas simbolu (iterēt rindu) var izmantot ciklu `for`:

```
for (int i = 0; str[i] != 0; i++)
    std::cout << str[i];
```

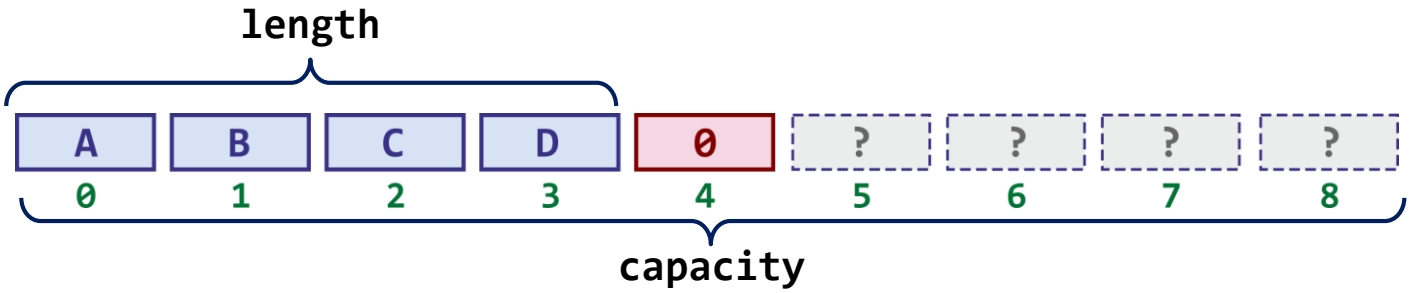
```
for (int i = 0; str[i] != 0; i++)
    std::cout << str[i];

for (int i = 0; i < str.length(); i++)
    std::cout << str[i];

for(char chr: str)
    std::cout << chr;
```

Rindas garums

Rindai tiek definēti divi izmēra jēdzieni – **ietilpība** (capacity) un **garums** (length). Ietilpība – tā ir rindas masīva izmērs. Garums – tās ir rindas simbolu skaits.



C style: `strlen(str)`

```
char str[8] = "ABCD";

strlen(str); // length 4

int len = 0;
for (int i = 0; str[i]; i++) len++;

sizeof(str); // capacity 8

strcpy_s(str, "012345"); // str = 012345
strlen(str); // length 6

strcpy_s(str, "0123456789"); // Error
```

C++ style: `str.length()`

```
std::string str = "ABCD";

str.length(); // length 4

str.capacity(); // capacity 15

str = "012345678901234567890";
str.length();   // length 20
str.capacity(); // capacity 31
```

Rindu piešķiršana

Vispopulārākā operācija ar mainīgiem, kas ļauj ierakstīt mainīgajā jauno vērtību – tas ir piešķiršana.

C style: `strcpy_s(dst, src)`

Rindas C stilā nevar piešķirt ar operāciju =, tā vietā ir jāizmanto `strcpy_s` funkcija.

```
char str[15] = "ABCD";

str = "01234"; // Error

strcpy_s(str, "01234"); // str = 01234

char src[] = "Helo";
strcpy_s(str, src); // str = Hello
```

C++ style: `dst = src`

Rindas C++ stilā var piešķirt ar operāciju =

```
std::string str = "ABCD";

str = "01234"; // str = 01234

std::string src = "Helo";
str = src; // str = Hello
```

Rindu konkatenācija

Konkatenācija – tā ir dažas rindas savienošana vienā rindā t.i. konkatenācija ļauj pierakstīt vienu rindu līdz otras beigām.

C style: `strcat_s(str1, str2)`

Rindas C stilā nevar savienot ar operāciju +, tā vietā ir jāizmanto `strcat_s` funkcija.

```
char str[10] = "ABCD";

str = str + "EF"; // Error

strcat_s(str, "EF"); // str = ABCDEF

char src[] = "GH";
strcat_s(str, src); // str = ABCDEFGH

strcat_s(str, "IJ"); // Error
```

C++ style: `str1 + str2`

Rindas C++ stilā var savienot ar operāciju +

```
std::string str = "ABCD";

str = str + "EF"; // str = ABCDEF

std::string src = "GH";
std::cout << str + src; // ABCDEFGH
```

Rindu salīdzināšana

Rindas gan C gan C++ stilos var salīdzināt, lai noskaidrotu vai tie satur vienādus simbolus. Izmantojot salīdzināšanu, ari var noskaidrot kāda no rindām ir lielāka vai mazāka nekā cita.

C style: `strcmp(str1, str2) == 0`

Rindas C stilā nevar salīdzināt ar salīdzināšanas operācijām (**==**, **<**, **>**...), tā vietā ir jāizmanto **`strcmp`** funkcija.

```
strcmp(str1, str2) = 0 → str1 == str2
strcmp(str1, str2) < 0 → str1 < str2
strcmp(str1, str2) > 0 → str1 > str2
```

```
char str[15] = "ABCD";

if (str == "ABCD")... // false

if (strcmp(str, "ABCD") == 0)... // true

char src[] = "abcd";
if (strcmp(str, src) == 0)... // false
```

C++ style: `str1 == str2`

Rindas C++ stilā var salīdzināt ar salīdzināšanas operācijām (**==**, **<**, **>**...)

```
str1 == str2      str1 != str2
str1 < str2       str1 <= str2
str1 > str2       str1 >= str2
```

```
std::string str = "ABCD";

if (str == "ABCD" )... // true

std::string src = "abcd";
if (str == src)... // false
```

Pamatoperācijas ar simbolu rindām

	C stye	C++ style
Virsraksts	`#include <cstring>`	`#include <string>`
Rindas garums	`strlen(str)`	`str.length()`
Piešķiršana	`strcpy_s(str, "new")`	`str = "new"`
Konkatenācija	`strcat_s(str, "add")`	`str += "add"`
Salīdzināšana	`strcmp(str, "sample") == 0`	`str == "sample"`
Apakšrinda	`strncpy_s(dst, str + 4, 3)`	`dst = str.substr(4, 3)`
Meklēšana	`strstr(str, "sample") != 0`	`str.find("sample") != std::string::npos`
Simbola meklēšana	`strchr(cstr, 'A') != 0`	`str.find('A') != std::string::npos`
Ielīkšana	`memmove(str + 7, str + 4, strlen(str) - 3);` `memmove(str + 4, "ins", 3);`	`str.insert(4, "ins")`
Dzēšana	`strcpy_s( str + 4, sizeof(str) - 4, str + 7 );`	`str.erase(4, 3)`

Rindu nodošana uz funkciju

Bieži vajag nodot funkcijai simbolu rindas kā formālie parametri. C stila rindas nodošanai var izmanto tikai references, bet C++ stila rindas var nodot gan kā references, gan kā vērtības.

C style: `char* str ≡ char str[]`

C stila rindas nodot funkcijai pēc tādiem pašiem principiem kā masīvus.

```
char s[100]{};

void func(char* str) {…}

func(s);
func("string"); // error

void func(const char* str) {…}

func(s);
func("string"); // OK
```

C++ style: `string str, string& str`

C++ stilā rindas nodošanai pielieto tādas pašas noteikumus, kā vienkāršiem datiem.

```
std::string s{};

void func(std::string str) {…}

func(s);
func("string");

void func(std::string& str) {…}

func(s);
func("string"); // error

void func(const std::string& str) {…}

func(s);
func("string"); // OK
```

Rindu atgriešana no funkcijas

Lai paņemtu rindas tipa rezultātu no funkcijas, C stilā pielieto formālus parametrus, bet C++ ļauj atgriezt rindas, kā funkcijas rezultātu.

C style: `void f(char* str)...`

C stilā rindas, tāpat kā masīvus, var dabūt no funkcijas tikai caur formālus parametrus.

```
void func(char* str)
{
    strcpy_s(str, 100, "string");
}
char s[100]{};
func(s);
std::cout << s;
```

C++ style: `std::string f()...`

C++ stilā rindas atgriešanai pielieto tādas pašas noteikumus, kā vienkāršiem datiem.

```
std::string func()
{
    return "string";
}
std::cout << func();
```

Tipa liešana

Nepieciešamības gadījumā var pārveidot rindas mainīga tipu starp C un C++ stilam.

C string → C++ string

```
char c_str[100] = "C string";
std::string cpp_str;

cpp_str = c_str;
std::cout << cpp_str << std::endl; // C string

std::cout << std::string(c_str).insert(1, "++"); // C++ string
```

C++ string → C string

```
char c_str[100];
std::string cpp_str = "C++ string";

strcpy_s(c_str, cpp_str.c_str());
std::cout << c_str << std::endl; // C++ string

strcpy_s(c_str, cpp_str.erase(1, 2).c_str());
std::cout << c_str << std::endl; // C string
```

Pārveidošana starp rindām un skaitļiem

string → integer

```c
char str[] = "1024";
int val = strtod(str, NULL);
std::cout << val * 2; // 2048
```

```c
std::string str = "1024";
int val = std::stoi(str);
std::cout << val * 2; // 2048
```

integer → string

```c
char str[100];
sprintf_s(str, "%d", 1024);
std::cout << str[0]; // 1
```

```c
std::string str;
str = std::to_string(1024);
std::cout << str[0]; // 1
```

string → float

```c
char str[] = "0.5";
float val = strtof(str, NULL);
std::cout << val * 2; // 1
```

```c
std::string str = "0.5";
float val = std::stof(str);
std::cout << val * 2; // 1
```

float → string

```c
char str[100];
sprintf_s(str, "%f", 0.5);
std::cout << str[0]; // 0
```

```c
std::string str;
str = std::to_string(0.5);
std::cout << str[0]; // 0
```