SCC-210
Algoritmos Avançados

Capítulo 3
Strings

João Luís G. Rosa

Strings & Códigos de Caracteres

- Caracteres são representados por códigos.
- Códigos de caracteres:
  - Mapeamento símbolo (em um dado alfabeto) $\leftrightarrow$ número
  - (Falta de) Padrão = (Falta de) Portabilidade
    - e.g. ASCII 10 (LF/NL) ou 13 (CR) para delimitar fim de linha de texto em arquivos de SOs diferentes.
- Código ASCII:
  - *American Standard Code for Information Interchange*
  - 7 bits – 128 símbolos.
  - Base dos tipos **char** de Pascal, C e C++ (bit mais significativo = zero)
- Variações e Extensões:
  - 8 bits – caracteres acentuados europeus (e.g. ISO Latin-1)
  - 16 bits – Unicode (Suportado por Java)

Código ASCII

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

Código ASCII

- Algumas propriedades úteis:
  - Região de caracteres não imprimíveis:
    - **[0 0 0 x x x x x]** – (0 a 31); ou
    - **[0 1 1 1 1 1 1 1]** – (127)
  - Tipos em seqüência:
    - Dígitos ('0' – '9'): 48 a 57
    - letras maiúsculas ('A' – 'Z'): 65 a 90 (1000001 a 1011010)
    - letras minúsculas ('a' – 'z'): 97 a 122 (1100001 a 1111010)
  - Rank do caractere em sua seqüência:
    - Rank('x') = ASCII('x') – ASCII('a') + 1;
    - Rank('X') = ASCII('X') – ASCII('A') + 1;
    - Rank('7') = ASCII('7') – ASCII('0') + 1;

Código ASCII

- Algumas propriedades úteis:
  - Conversão maiúscula – minúscula:
    - 'x' = 'a' + ('X' – 'A')
    - 'x' = 'X' | 0x20 (00100000)
  - Conversão minúscula – maiúscula:
    - 'X' = 'A' + ('x' – 'a')
    - 'X' = 'x' & 0xDF (11011111)
  - Precaução em ordenação por código:
    - ASCII('B') < ASCII('a')

Strings

- Tipos de Representação:
  - Arranjos com terminação nula:
    - C
    - Último elemento do vetor é '\0'
    - Espaço para o nulo deve sempre ser previsto !
  - Arranjo com tamanho:
    - Primeira célula armazena o comprimento da string
    - Elimina necessidade de caractere de terminação
    - Objetos String Java e Pascal (representação interna)
  - Lista encadeada de caracteres:
    - Pode acelerar procedimentos de sucessivas inserções e remoções de substrings em uma string
    - Normalmente evitado por requerer ponteiros para cada caractere

Bibliotecas C (Algumas Funções)

```
#include <ctype.h>    /* include the character library */

int isalpha(int c);   /* true if c is either upper or lower case */
int isupper(int c);   /* true if c is upper case */
int islower(int c);   /* true if c is lower case */
int isdigit(int c);   /* true if c is a numerical digit (0-9) */
int ispunct(int c);   /* true if c is a punctuation symbol */
int isxdigit(int c);  /* true if c is a hexadecimal digit (0-9,A-F) */
int isprint(int c);   /* true if c is any printable character */
int toupper(int c);   /* convert c to upper case -- no error checking */
int tolower(int c);   /* convert c to lower case -- no error checking */


#include <string.h>   /* include the string library */

char   *strcat(char *dst, const char *src);     /* concatenation */
int     strcmp(const char *s1, const char *s2); /* is s1 == s2? */
char   *strcpy(char *dst, const char *src);     /* copy src to dst */
int     strlen(const char *s);                  /* length of string */
char   *strstr(const char *s1, const char *s2)  /* search for s2 in s1 */
```

Conversão entre tipos

- Conversão tipos diversos para string:
  - **`int sprintf(char * str, const char * format, ...)`**
    - O mesmo funcionamento de **`printf`** mas com a saída armazenada em **`str`**.
    - Ex.: **`sprintf(buffer, "%d %5.2f %c", 21, 2.1, 'A');`**
  - **`int sscanf (const char * str, const char * format, ...);`**
    - O mesmo funcionamento de **`scanf`**, mas com a entrada dos dados a partir de **`str`**;
    - Ex.: **`sscanf(buffer, "%d %d", &a, &b);`**
    - //buffer = "433 1211"

Strings em C++

- C++ possui uma classe string que é mais simples de trabalhar do que os strings (vetores de caracteres) em C;
- A classe string provê uma gama de métodos e operadores (=, ==, +) que fazem com que a classe se pareça com um tipo de dados fundamental;
- Não há necessidade em se preocupar com o tamanho do string.

Strings em C++

- Dois exemplos vão nos ajudar a entender algumas das principais funcionalidades da classe string.
- Os exemplos foram adaptados de Josuttis (1999). Maiores detalhes podem ser encontrados em:
  - Josuttis N. The C++ Standard Library. Addison Wesley, 1999.
  - **`http://www.sgi.com/tech/stl/`**

Strings em C++: Exemplo 1

- Primeiro exemplo, programa que gera nomes de arquivos temporários:
  - `string1 prog.dat mydir hello. oops.tmp end.dat`
- Saída:
  - `prog.dat => prog.tmp`
  - `mydir => mydir.tmp`
  - `hello. => hello.tmp`
  - `oops.tmp => oops.xxx`
  - `end.dat => end.tmp`

Strings em C++: Exemplo 1

```
#include<cstdio>
#include<string>

using namespace std;

int main(int argc, char* argv[]) {
    string filename, basename, extname, tmpname;
    const string suffix("tmp");

    for (int i=1; i<argc; i++) {
        filename = argv[i];

        string::size_type idx = filename.find('.');
        if (idx == string::npos)
            tmpname = filename + '.' + suffix;
        else {
            basename = filename.substr(0, idx);
            extname = filename.substr(idx+1);
            if (extname.empty()) {
                tmpname = filename;
                tmpname += suffix;
            }
```

(Josuttis, 1999)

Strings em C++: Exemplo 1

```
            else if (extname == suffix) {
                tmpname = filename;
                tmpname.replace(idx+1, extname.size(), "xxx");
            } else {
                tmpname = filename;
                tmpname.replace(idx+1, string::npos, suffix);
            }
        }
        printf("%s => %s\n", filename.c_str(), tmpname.c_str());
    }
    system("pause");
}
```

(Josuttis, 1999)

Strings em C++: Exemplo 2

- No segundo exemplo, o programa lê linhas da entrada padrão, extrai palavras e as imprime em ordem inversa:
- Exemplo:
  - frase teste
  - esarf etset

Strings em C++: Exemplo 2

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

using namespace std;

int main(int argc, char* argv[]) {
  const string delims(" \t,.");
  string line;

  while (getline(cin, line)) {
        string::size_type begIdx, endIdx;

        begIdx = line.find_first_not_of(delims);

        while (begIdx != string::npos) {
              endIdx = line.find_first_of(delims, begIdx);
              if (endIdx == string::npos)
                 endIdx = line.length();
              for (int i = endIdx-1; i >= static_cast<int>(begIdx); i--)
                  cout << line[i];
              cout << ' ';

              begIdx = line.find_first_not_of(delims, endIdx);
        }
        cout << endl;
  }
}
```

(Josuttis, 1999)

Entrada/Saída em C++

- **`#include<iostream>`**
- *Standard input stream* (**`cin`**) e *standard output stream* (**`cout`**)
  - **`cin >> str;  // Similar scanf`**
  - **`cout << str;`**
  - **`cout << str << "+" << i << endl;`**
- Função **`getline`**
  - **`getline(cin, str); // Similar gets ou fgets`**
  - **`getline(cin, str, "-");`**

Referências

- Batista, G. & Campello, R.
  - Slides disciplina *Algoritmos Avançados*, ICMC-USP, 2007.
- Skiena, S. S. & Revilla, M. A.
  - *Programming Challenges – The Programming Contest Training Manual*. Springer, 2003.