STRING


Pemrograman
C++
Dasar
PHP
C++
Ruby
C
C++
Pascal
Java
C++
Program
Python
Python
PL-SQL

What is a String in C ?

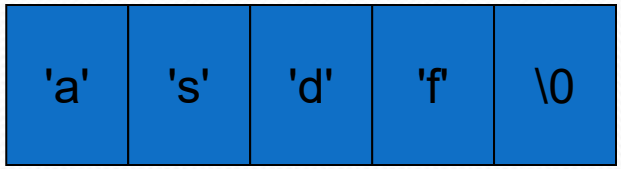
- *Arrays :* data structures that contain a *collection* of data objects of the *same data type*
- Strings are really just arrays of characters
- A string is an *array of characters terminated by the **NUL** character*
  - Data type is *array of characters*
  - The **NUL** character is a character with a *numeric value* of zero
    - In C, it is represented by the escape sequence, \0 (NUL)
- String constant (or string literal)
  - Any series of characters enclosed in double quotes
    - Ex. "Hello world"
- String example program
  - stores a string constant
  - stores individual elements
    - note: character constants
    - note: terminate with '\0‘ to make the character array a string

Strings as zero terminated array

- In C a string is a zero-terminated array of characters

char s[ ] = "asdf";

S :



in C, the convention is to store a string as a char array, terminated with '\0'

char str[] = "Dave";

What is a String?

strings1.c

```
/* strings1.c */
#include <stdio.h>
int main()
{
 int i;
 char str1[] = "Hello world";
 char str2[5];
 str2[0]='M';
 str2[1]='E';
 str2[2]='3';
 str2[3]='0';
 str2[4]='\0';
 printf("str1==%s\n",str1);
 for(i=0; i<5; i++)
 {
   printf("str2[%d]==%c\n",i,str2[i]);
 }
 return 0;
}
```

Initializing String Variables

- Declare as an array of type **char**
  - Can initialize *implicitly*
  - or *explicitly*

    (using a string constant or using individual characters and a terminating NUL character)
    - If explicit, make sure that:

$$n_{elements} \geq n_{chars\ in\ string} + 1$$

```
/* strings1.c */
#include <stdio.h>
int main()
{
 int i;
 char str1[] = "Hello world";
 char str3[12] = "Hello world";
 char str2[5];
 str2[0]='M';
 str2[1]='E';
 str2[2]='3';
 str2[3]='0';
 str2[4]='\0';
 printf("str1==%s\n",str1);
 for(i=0; i<5; i++)
 {
   printf("str2[%d]==%c\n",i,str2[i]);
 }
 return 0;
}
```

American Standard Code for Information Interchange (ASCII) Table

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

Printing Strings to stdout

- Can print an entire string using printf and %s format specification
- Can print individual elements of a string by indexing and using %c format specification

```
/* strings1.c */
#include <stdio.h>
int main()
{
 int i;
 char str1[] = "Hello world";
 char str3[12] = "Hello world";
 char str2[5];
 str2[0]='M';
 str2[1]='E';
 str2[2]='3';
 str2[3]='0';
 str2[4]='\0';
 printf("str1==%s\n",str1);
 for(i=0; i<5; i++)
 {
   printf("str2[%d]==%c\n",i,str2[i]);
 }
 return 0;
}
```

Manipulating Strings in C

- No native support for strings in C
- Use **The Standard Function C Library** for string
  - #include <string.h>

Standard function libraries

<stdio.h>	**printf()**, **scanf()**, etc.
<string.h>	**strcmp()**, etc.
<ctype.c>	**isspace()**, etc.
<stdlib.h>	**malloc()**, etc.
<math.h>	**sqrt()**, etc.

Copying and Concatenating Strings

- to ***copy*** str2 to str1 (order resembles *assignment*):
  - **strcpy(str1, str2);**
    - **str1 = str2; /* Will NOT work!! */**

- to ***add*** (*concatenate*) str2 to the end of str1:
  - **strcat(str1, str2);**
    - returns the value of str1 with str2 added to the end
      - Note: it clobbers the /0 at the end of str1
      - Make sure that str1 has room for str2

Comparing strings

if (strcmp(s1,s2) = = 0) { /* *do* ***s1*** *and* ***s2*** *hold the same characters?* */
}

Finding the length of a string

- Use strlen(string)

  returns length of the string
- strlen() is a predefined function in C whose definition is contained in the header file “string.h”.
- strlen() accepts a pointer to an array as argument and walks through memory at run time from the address we give it looking for a **NULL** character and counts up how many memory locations it passed before it finds one.
- The main task of strlen() is to count the length of an array or string.

sizeof()

- Sizeof operator is a compile time unary operator which can be used to compute the size of its operand.
- The result of sizeof is of unsigned integral type which is usually denoted by size_t.
- sizeof can be applied to any data-type, including primitive types such as integer and floating-point types, pointer types, or compound datatypes such as Structure, union etc.

Strings – Practice

- Declare:
  - A string variable named **me30**, and initialize it to hold the string, "ME 30 rocks!"
- Determine
  - The minimum number of character array elements needed to store the string
  - length of the string
  - What is stored in **me30[2]?**
  - What is stored in **me30[11]?**

```c
#include <stdio.h>
#include <string.h>

int main ()
{

char me30[]="ME 30 rocks!";

int len1, len2, len3,len4 ;
len1 = strlen(me30);
printf("strlen(me30) : %d\n",len1);
len2 = sizeof(me30);
printf("size of (me30) : %d\n",len2);
len3 = strlen(me30)+1;
printf("strlen(me30)+1 : %d\n",len3);
len4 = sizeof(me30)/sizeof(char);
printf("sizeof(me30)/sizeof(char) : %d\n",len4);
printf("char in me30[2] : %c\n",me30[2]);
printf("char in me30[11] : %c\n",me30[11]);
printf("char in me30[10] : %c\n",me30[10]);
return 0;
}
```

Output

```
strlen(me30) : 12
size of (me30) : 13
strlen(me30)+1 : 13
sizeof(me30)/sizeof(char) : 13
char in me30[2] :
char in me30[11] : !
char in me30[10] : s
```

```
#include <stdio.h>
#include <string.h>

int main ()
{
char str1[10] = "Hello";
char str2[10] = "World";
char str3[10];
int len ;

// copy str1 into str3
strcpy( str3, str1);
printf("strcpy( str3, str1) : %s\n",str3);

// concatenates str1 and str2
strcat( str1, str2);
printf("strcat( str1, str2) : %s\n",str1);

// total lenghth of str1 after concatenation
len = strlen(str1);
printf("strlen(str1) : %d\n",len);

return 0;
}
```

Output

*** buffer overflow detected ***: ./prog terminated

```
char str1[11] = "Hello";
```

Output

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

Fundamentals of Characters and Strings

- String
  - Series of characters treated as one unit
  - Can include letters, digits, special characters **+**, **–**, *****
  - String literal (string constants)
    - Enclosed in double quotes, for example:

      "I like C++"
  - Can also reference as an entire string
  - Using functions found in the <cstring> library

```
if (strcmp (s, "hello") == 0){...}
if (strlen (s) == 1){...}
```

Diganti
#include <string> apa yang akan terjadi ?

```
#include <iostream>
#include <cstring>
using namespace std;
int main ()
{
 char str1[11] = "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;
}
```

```
#include <iostream>
#include <string.h>
using namespace std;
int main ()
{
 char str1[11] = "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;
}
```

C ++ Strings

- Not necessarily **`null`** terminated
- **`string`** is not a pointer, but a class
- Many member functions take start position and length
  - If length argument too large, max chosen

C ++ Strings

Creating String Objects

```
#include <string>
//string initialization

string s;  //s contains 0 characters
string s1( "Hello" );  //s1 contains 5 characters
string s2 = “Hello”; //s2 contains 5 characters
                     //implicitly calls the constructor

string s3( 8, 'x' );   //s3 contains 8  'x' characters
string s4 = s3;        //s4 contains 8  'x' characters

string s5(s2, 3, 2);  //s5 copies a substring of s2; it contains ”lo”
```

string type in the **<string>** header file.

string_characteristics.cpp

C ++ Strings

String Objects

The C++ string class also defines a length() function for extracting how many characters are stored in a string.

```
cout << s.length() << endl;
```

Prints 4 for the string s == “Leon”

You can also use the ***subscript operator*** **[ ]** to access individual characters:

e.g. **s[0] = ‘N’ ;** *//where* **index: 0** to **length-1**

```
#include <iostream>
#include <string>
using namespace std;
int main ()
{
 string str1 = "Hello";
 string str2 = "World";
 string str3;
 int len1, len2 ;
 // 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
 len1 = str3.size();
 cout << "str3.size() : " << len1 << endl;
 len2 = str3.length();
 cout << "str3.length() : " << len2 << endl;
 return 0;
}
```

Output

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

C ++ Strings

String Objects

You can also concatenate C++ strings using the + and += operators:

```
string s = "ABCD*FG";
string s2 = "Robot";
string s5 = "Soccer";
string s6  = s + "HIJK";   //changes s6 to "ABCD*FGHIJK

s2 += s5;  //changes s2 to "RobotSoccer"
```

Concatenation

- **Concatenation**
  - `s3.append( "pet" );`
  - `s3 += "pet";`
    - Both add `"pet"` to end of `s3`
  - `s3.append( s1, start, N );`
    - Appends `N` characters from `s1`, beginning at index `start`

Inserting Characters into a string

- **`s1.insert( index, s2 )`**
  - Inserts **`s2`** before position **`index`**
- **`s1.insert( index, s2, index2, N );`**
  - Inserts substring of **`s2`** before position **`index`**
  - Substring is **`N`** characters, starting at **`index2`**

String Objects

C++ strings can be compared using relational operators just like fundamental types:

```
If(s2 < s5)
    cout << “s2 lexicographically precedes s5 \n”;

while(s4==s3) //…
```

String Objects

Substring function: substr()

```
s6 = "ABCD*FGHIJK";
s4 = s6.substr(5, 3); //changes s4 to "FGH"
```

s4 gets a substring of s6, starting at index 5 and taking 3 characters

String Objects

erase() and replace() functions:

```
s6 = "ABCD*FGHIJK";
s6.erase(4, 2); //changes s6 to "ABCDGHIJK";

s6.replace(5, 2, "xyz"); //changes s6 to "ABCDGxyzJK";
```

replace 2 characters from s6, starting at index 5, with "xyz"

String Objects

find() function

returns the index of the first occurrence of a given substring:

```
string s7 = "Mississippi River basin";   //23 characters
cout << s7.find("si") << endl;   //prints 3
cout << s7.find("so") << endl;  //prints 23, the length of the string
```

If the find() function fails, it returns the length of the string it was searching.

i.e. find() returns 4,294,967,295

Assignment

- **Assignment**
  - `s2 = s1;`
    - Makes a separate copy
  - `s2.assign(s1);`
    - Same as `s2 = s1;`
  - `myString.assign(s, start, N);`
    - Copies `N` characters from `s`, beginning at index `start`
  - Individual character assignment
    - `s2[0] = s3[2];`

Range-checking

- **Range-checking**
  - **`s3.at( index );`**
    - Returns character at **`index`**
    - Can throw an **`out_of_range`** exception
  - `[]` has no range checking

```
#include <exception>
...

string s = "leon";
try{
   char letter = s.at( 50 );
   cout <<"letter is = " << letter << endl;
}

catch(exception&  e){
            cout << "out_of_range exception:  " <<  e.what() <<  endl;
}
```

Comparing strings

- Overloaded operators
  - ==, !=, <, >, <= and >=
  - returns `bool`
- `s1.compare(s2)`
  - returns positive if `s1` is lexicographically greater
    - compares letter by letter
    - `'B'` lexicographically greater than `'A'`
    - `'a'` lexicographically greater than `'A'`
    - `'a'` lexicographically greater than `'Z'`
  - returns negative if less; zero if equal
    - `Sample order: 'A',"Apple", "Banana", "Zest", 'a', "apricot", "leon"`
  - `s1.compare(start, length, s2, start, length)`
    - Compare portions of `s1` and `s2`
  - `s1.compare(start, length, s2)`
    - Compare portion of `s1` with all of `s2`

Compare.cpp

Substrings

- Function **`substr`** gets a substring
  - **`s1.substr( start, N );`**
  - gets **`N`** characters, beginning with index **`start`**
  - returns substring

Finding Strings and Characters in a string

- **Find functions**
  - If found, **index** returned
  - If not found, **`string::npos`** returned
    - Public static constant in class string
  - **`s1.find( s2 )`**
  - **`s1.rfind( s2 )`**
    - Searches right-to-left
  - **`s1.find_first_of( s2 )`**
    - Returns first occurrence of any character in **`s2`**
    - **`Example: s1.find_first_of( "abcd" )`**
      - Returns index of first **`'a'`**, **`'b'`**, **`'c'`** or **`'d'`**

Finding Strings and Characters in a string

- **Find functions**
  - `s1.find_last_of( s2 )`
    - Finds last occurrence of any character in `s2`
  - `s1.find_first_not_of( s2 )`
    - Finds first character NOT in `s2`
  - `s1.find_last_not_of( s2 )`
    - Finds last character NOT in `s2`

Replacing Characters in a string

- **`s1.erase( start )`**
  - Erase from index **`start`** to end of string, including **`start`**
- Replace
  - **`s1.replace( begin, N, s2)`**
    - **`begin`**: index in **`s1`** to start replacing
    - **`N`**: number of characters to replace
    - **`s2`**: replacement string
  - **`s1.replace( begin, N, s2, index, num )`**
    - **`index`**: element in **`s2`** where replacement comes from
    - **`num`**: number of elements to use when replacing
  - Replace can overwrite characters

Example

`s1.replace( begin, N, s2, index, num )`

- **`begin`**: index in **`s1`** to start replacing
- **`N`**: number of characters to replace
- **`s2`**: replacement string
- **`index`**: element in **`s2`** where replacement comes from
- **`num`**: number of elements to use when replacing

```
string str = "this is an example string.";
string str3="sample phrase";

str.replace(19,6, str3, 7, 6);    // "this is an example phrase."
```

String Objects

C++ strings can be converted to C-strings:

```
string s = "ABCDEFG";
const char* cs = s.c_str();
```

Converts s into the C-string cs.

The c_str() function has a return type const char*

Conversion to C-Style char*

- **Conversion functions**
  - **`String`**s are not necessarily null-terminated
  - **`s1.copy( ptr, N, index )`**
    - Copies **`N`** characters **into** the array **`ptr`**
    - Starts at location **`index`**
    - Need to null terminate

```
char str[8];

string s2 = "cathode";

s2.copy(str, 5, 2); //copy 5 characters into str
                    //starting at index 2

//strcat(str,"\0"); //does not work
str[5] = '\0';      //this is required

cout << "str = " << str << endl;
cout << "s2 = " << s2 << endl;
```

Output:

str = thode
s2 = cathode

Conversion to C-Style char * Strings

- **Conversion functions**
  - **`s1.c_str()`**
    - Returns **`const char *`**
    - Null terminated
    - *e.g.* Useful for filenames:
      ifstream in(**s1.c_str()**);
  - **`s1.data()`**
    - Returns **`const char *`**
    - NOT null-terminated

Warning!

- No conversion from **`int`** or **`char`**
  - The following definitions could return errors, or warnings only, but then would cause the program to crash afterwards
    - `string error1 = 'c';`
    - `string error2( 'u' );`
    - `string error3 = 22;`
    - `string error4( 8 );`
  - However, it can be assigned one **`char`** after its declaration:
    - `s = 'n';`

cin and strings

- The extraction operator can be used on cin to get strings of characters in the same way as with fundamental data types:

```
string mystring;
cin >> mystring;
```

- However, cin extraction always considers spaces (whitespaces, tabs, new-line...) as terminating the value being extracted, and thus extracting a string means to always extract a single word, not a phrase or an entire sentence.
  To get an entire line from cin, there exists a function, called getline, that takes the stream (cin) as first argument, and the string variable as second. For example:

```
// cin with strings
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystr;
  cout << "What's your name? ";
  getline (cin, mystr);
  cout << "Hello " << mystr << ".\n";
  cout << "What is your favorite team? ";
  getline (cin, mystr);
  cout << "I like " << mystr << " too!\n";
  return 0;
}
```

```
What's your name? Homer Simpson
Hello Homer Simpson.
What is your favorite team? The Isotopes
I like The Isotopes too!
```

Notice how in both calls to getline, we used the same string identifier (mystr). What the program
does in the second call is simply replace the previous content with the new one that is introduced.

STRING INPUT AND OUTPUT

C++ Routine	Description
cout	String output to the screen
cin	String input from the keyboard, but cin does not input a space character. It stops input a string when it reads a space character.
cin.getline(str, length,char)	String input from the keyboard. cin.getline( ) read a space character.
	str - a string of character pointer or a character array. length - an integer constant or a variable indicating the maximum number of input characters including the null character. char - an optional character constant or variable specifying the terminating character. If this optional third argument is omitted, the default terminating character is the newline (\n) character. Pressing the enter key generates a newline character. A statement such as cin.getline(message,80,'x') will stop accepting characters whenever the x key is pressed.
chr = cin.get( )	Input single character from the keyboard. chr - a character constant variable

Example	Output

#include<iostream>
using namespace std;
int main( )
{char message1[80];
char *message2;
cout <<"Enter a string for message1: \n";
cin.getline(message1,80);
cout << "Enter a string for message2: \n";
cin.getline(message2,80);
cout <<message1<< " and " <<message2;
}
	Enter a string for message1:
	Good morning
	Enter a string for message2:
	have a nice day
	Good morning and have a nice day

CHARACTER STRING FUNCTIONS

- C++ provides several functions that allow you to test and manipulate character data.
  The function prototypes are found in the header file name <ctype.h>. Remember to
  add the line #include <ctype.h> in program that use these functions. The table below
  lists and describes the character functions.
- Each function expects one integer
  argument - the ASCII value of the character to be tested. Each function returns a nonzero value (true) if the condition tested is true and 0 (false) if the condition tested is false.

C++ Functions	Description
isalpha(character)	Returns a nonzero number if the character is a letter ('A' - 'Z', 'a' - 'z'); otherwise it returns zero.
isalnum(character)	Returns a nonzero number if the character is a letter ('A' - 'Z', 'a' - 'z', or '0' - '9'; otherwise it returns zero.
isdigit(character)	Returns a nonzero number if the character is digit (0 through 9); otherwise it returns a zero.
isspace(character)	Returns a nonzero number if the character is a whitespace (tab, space, newline); otherwise it returns a zero.
isupper(character)	Returns a nonzero number if the character is uppercase; otherwise it returns a zero.
islower(character)	Returns a nonzero number if the character is lowercase; otherwise it returns a zero.
toupper(character)	Return the uppercase equivalent if the character is lowercase; otherwise it returns the character unchanged.
tolower(character)	Return the lowercase equivalent if the character is uppercase; otherwise it returns the character unchanged.

C++ Functions	Description
toupper(character)	Return the uppercase equivalent if the character is lowercase; otherwise it returns the character unchanged.
tolower(character)	Return the lowercase equivalent if the character is uppercase; otherwise it returns the character unchanged.
atoi(string)	Converts an ASCII string to an integer (include #<stdlib.h> in your program)
atof(string)	Converts an ASCII string to an float (include #<stdlib.h> in your program)

Example

```
#include<iostream>
#include<string>
#include<cctype>
using namespace std;
int main( )
{ char name[20];
cout<<"Enter your name:\n ";
cin.getline(name,20);
for( int i = 0; i < strlen(name) ; i++)
{ if (islower(name[i]) )
//convert to uppercase
name[i] = toupper(name[i]);
else
//convert to lowercase
name[i] = tolower(name[i]);
}
//Display the result
cout << "The conversion is:\n";
cout << name << endl;
}
```

Output

```
Enter your name:
Phuong D. Nguyen
The conversion is:
pHUONG d. nGUYEN
```

Example

```
#include<iostream>
#include<string>
#include<cctype>
using namespace std;
int main()
{ char *name;
cout<<"Enter your name:\n ";
cin.getline(name,20);
for( int i = 0; i < strlen(name) ; i++)
{ if (islower(*(name + i) ) )
//convert to uppercase
*( name + i) = toupper(*(name + i));
else
//convert to lowercase
*( name + i) = tolower(*(name + i));
}
//Display the result
cout << "The conversion is:\n";
cout << name << endl;
}
```

Output

```
Enter your name:
Phuong D. Nguyen
The conversion is:
pHUONG d. nGUYEN
```

String Stream Processing

- allows a string to be used as an internal file
- useful for buffering input and output

- I/O of strings to and from memory
  - Called in-memory I/O or string stream processing
  - Classes
    - **`istringstream`** // input from string
    - **`ostringstream`** // output to a string
    - **stringstream(string)** // most useful
    - **Requires** **<sstream>** and **<iostream>** headers
  - Use string formatting to save data to memory

Output String Stream

```
ostringstream  oss;

int n = 44;
float x = 3.14;

oss << "Hello!\t" << n << '\t' << x;
string  s = oss.str();

cout << endl << s << endl;
```

Serves as a conduit to an anonymous string which can be read with the built-in oss.str() function that is bound to the oss object

Remember sprintf()?, how does it compare to this one?

output_string_stream.cpp

Input String Stream

```
const string buffer = oss.str();
istringstream  iss(buffer);

string word;
int m;
float y;

iss >> word >> m >> y;

s = iss.str();
cout << endl << s << endl;

cout << "word = " << word << endl;
cout << "m = " << m << endl;
cout << "y = " << y << endl;
```

iss is defined and bound to buffer

Contents of **buffer** can be accessed as elements of a string, or by formatted input through the **iss** object.

Remember sscanf()?, how does it compare to this one?

input_string_stream.cpp

All extractions from iss will come from the contents of buffer, as if it were an external file.

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include <sstream>
using namespace std;
int main(){

  string s1("mydata.txt");
  ifstream in( s1.c_str() );

  char buffer[1024];
  while( in.getline( buffer, 1024 ) ){

    string stemp( buffer );
    cout << "Line is:" << stemp << endl;

    if( stemp[0] != '#' ){
      stringstream stris( stemp );
      double d1, d2;
      stris >> d1 >> d2;
      cout << d1 << "," << d2 << endl;
    }
    cout << endl;
  }
  in.close();
  return 0;
}
```

Using string example

- **Input file:**

```
1.0   2.0
1.1    2.4
1.8  2.8
#1.34  2.99
1.4     8.99
```

- **Example Output:**

```
Line is:1.0   2.0
1,2

Line is:1.1    2.4
1.1,2.4

Line is:1.8  2.8
1.8,2.8

Line is:#1.34  2.99

Line is:1.4     8.99
1.4,8.99
```

(or could use strtok, C String tokenizers)

Alternatively: (no C-style char*)

File_stringstream2.cpp

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include <sstream>
using namespace std;
int main(){

  string s1("mydata.txt");
  ifstream in( s1.c_str() );

  char buffer[1024];
  while( in.getline( buffer, 1024 ) ){

    string stemp( buffer );
    cout << "Line is:" << stemp << endl;

    if( stemp[0] != '#' ){
      stringstream stris( stemp );
      double d1, d2;
      stris >> d1 >> d2;
      cout << d1 << "," << d2 << endl;
    }
    cout << endl;
  }
  in.close();
  return 0;
}
```

```
int main(){

  string s1("mydata.txt");
  ifstream in( s1.c_str() );

  string buffer;
  while(getline( in, buffer ) ){


    cout << "Line is:" << buffer << endl;

    if( buffer[0]  !=  '#' ){
      istringstream stris( buffer );
      double d1, d2;
      stris >> d1 >> d2;
      cout << "data: " << d1 << "," << d2 << endl;
    }
    cout << endl;
  }
  in.close();
  return 0;
}
```

Summary

C++ strings are safer and easier to use than C string.

Next Templates

Method	Use

append(char *pt);
append(char *pt, size_t count);
append(string &str, size_t offset,size_t count);
append(string &str);
append(size_t count, char ch);
append(InputIterator Start, InputIterator End);
	Appends characters to a string from C-style strings, char's or other string objects.

at(size_t offset);	Returns a reference to the character at the specified position. Differs from the subscript operator, [], in that bounds are checked.
begin();	Returns an iterator to the start of the string.
*c_str();	Returns a pointer to C-style string version of the contents of the string.
clear();	Erases the entire string.
copy(char *cstring, size_t count, size_t offset);	Copies "count" characters into a C-style string starting at offset.
empty();	Test whether a string is empty.
end();	Returns an iterator to one past the end of the string.

erase(iterator first, iterator last);
erase(iterator it);
erase(size_t pos, size_t count);
	Erases characters from the specified positions.

Method	Use

find(char ch,size_t offset = 0);
find(char *pt,size_t offset = 0);
find(string &str,size_t offset = 0);
	Returns the index of the first character of the substring when found. Otherwise, the special value "npos" is returned.

find_first_not_of();	Same sets of arguments as find. Finds the index of the first character that is not in the search string.
find_first_of();	Same sets of arguments as find. Finds the index of the first character that is in the search string.
find_last_not_of();	Same sets of arguments as find. Finds the index of the last character that is not in the search string.
find_last_of();	Same sets of arguments as find. Finds the index of the last character that is in the search string.

insert(size_t pos, char *ptr);
insert(size_t pos, string &str);
insert(size_t pos, size_t count, char ch);
insert(iterator it, InputIterator start, InputIterator end);
	Inserts characters at the specified position.

push_back(char ch);	Inserts a character at the end of the string.

replace(size_t pos, size_t count, char *pt);
replace(size_t pos, size_t count, string &str);
replace(iterator first, iterator last, char *pt);
replace(iterator first, iterator last, string &str);
	Replaces elements in a string with the specified characters. The range can be specified by a start position and a number of elements to replace, or by using iterators.

size();	Returns the number of elements in a string.
swap(string &str);	Swaps two strings.

Questions?