

CS2311 Computer Programming

Character and String

Character data type

- Written between **single quotes**, length must be one:
'A', 'b', '*'
- In C++, a `char` type variable is internally represented as a 8-bit integer, according to the ASCII table
- Therefore a `char` can also be used & printed as integer (*with type casting*)
- Example:

```
cout << 'a'; /* a is printed */  
cout << (int)'a'; /* 97 is printed */  
cout << (char)97; /* a is printed */
```

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		DEL

Example: ASCII value of character '7' is 55 but not 7!

char features

As mentioned in Lec 2, char can also perform arithmetic and comparison

A	B	...	Z	...	a	b	...	z
65	66	...	90	..	97	98	...	122

'a' + 1 has the value of 'b',
'b' + 1 has the value of 'c', ...

test if character c is lower-case letter:

```
if ('a' <= c && c <= 'z') ... (recommended)
```

same as:

```
if (97 <= c && c <= 122) ... (not recommended)
```

If the variable c has the value of a lowercase letter, then the expression

`c + 'A' - 'a' /* same as c + (65 - 97) */`

has the value of the corresponding uppercase letter

Example – Print the ASCII code of a character

```
#include <iostream>
using namespace std;
void main() {
    char c;
    cout<<"Input a character: ";
    cin>>c;
    int num = (int)c;    //type casting
    cout<<"The ASCII code is "<<num;
}
```

Example – Conversion between upper/lower cases

```
#include <iostream>
using namespace std;
void main(){
    char c1='D', c2='e', c3,c4;
    c3=c1+'a'-'A'; //convert to lowercase
    c4=c2+'A'-'a'; //convert to uppercase
    cout<< "c3=" << c3 << ", c4=" << c4 << endl;
    //c3=d, c4=E
}
```

Reading a *non-space* character

```
char c1;  
cin >> c1;
```

When `cin >> c1` is executed, the program will ask the user for input (*skipping all <space>/<enter> characters*)

Q: What if the user inputs more than 1 character ?

A: The extra characters are stored in the input *buffer* in memory. When later `>>` are encountered, content from the buffer is returned directly without waiting for user input.

Some facts about keyboard Input

```
char c1,c2,c3;  
cin >> c1; //enter the string "CS2311", c1 get 'C'  
cin >> c2; //c2 get the character 'S' from buffer  
cin >> c3; //c3 get the character '2' from buffer
```

	c1	c2	c3	Input buffer
(user input "cs2311")				C S 2 3 1 1 ↵
cin >> c1;	'C'			S 2 3 1 1 ↵
cin >> c2;	'C'	'S'		2 3 1 1 ↵
cin >> c3;	'C'	'S'	'2'	3 1 1 ↵

Example

- Write a program which reads a character from the user and output the character type.
- The program should distinguish between the following types of characters
 - An upper case character (' A ' – ' Z ')
 - A lower case character (' a ' – ' z ')
 - A digit (' 0 ' – ' 9 ')
 - Special character (e.g. ' # ' , ' \$ ' , etc)

Exercise – Fill in the missing parts

```
#include<iostream>
using namespace std;
void main(){
    char c;
    cin >> c;
    if (_____)
        cout << "An upper case character\n";
    else if (_____)
        cout << "A lower case character\n";
    else if (_____)
        cout << "A digit\n";
    else
        cout << "Special character\n";
}
```

Answer

```
#include <iostream>
using namespace std;
void main(){
    char c;
    cin >> c;
    if (c>='A' && c<='Z')
        cout << "An upper case character\n";
    else if (c>='a' && c<='z')
        cout << "A lower case character\n";
    else if (c>='0' && c<='9')
        cout << "A digit\n";
    else
        cout << "Special character\n";
}
```

IMPORTANT: Note the difference between '0' and 0

Read *any* character with `cin.get`

- `cin.get()`: member function of `cin` to read in one character from input
- Unlike `cin>>` , `cin.get()` WON'T skip over white spaces (i.e. you can use it to read <space> and <enter>)
- Syntax:

```
char c;  
cin.get(c);
```

Read *any* character with `cin.get`

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

void main() {
    char c;
    do {
        cin.get(c);
        cout << c;
    } while (c != '\n');
}
```

String -- as character array

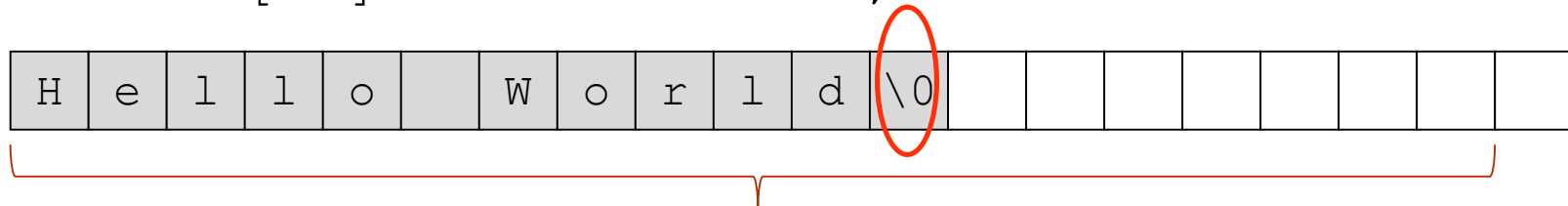
cstring vs string object

- In C++, there are two types of strings:
- As an array of `char` (*the same mechanism as the C language*), also known as the so-called *cstring*
- `std::string` class defined in C++ `<string>` library (*not covered in this course*)

String : cstring

- A cstring is a `char` array, which is terminated by the **end-of-string mark** (also called **null character**) `'\0'` (*ASCII code: 0*) .
- In other words, whatever enclosed by double quotation marks `".."` will have `'\0'` *automatically* added to the end
- Because you need `'\0'`, so a character array of size **n** may stores only strings with a maximum length of **n-1**
- Consider the definition

```
char str[20]="Hello World";
```



This character array may store a string with maximum length of 19

Rule of Safety

- Declare a string with at least one character more than needed
- E.g.
 - `char X[4]="ABC";` //Correct
 - `char Y[3]="ABC";` //Wrong, 3 is not enough!

Double quotation mark for **string**

`char Y[3]="ABC";` is actually IDENTICAL to..

`char Y[3] = {'A','B','C','\0'};`

Single quotation mark for **character**

Obviously a [3] array cannot store 4 characters!

cstring: Declaration and Initialization

- String variable can be declared in one of the two ways:
- Without initialization
 - `char identifier[required size+1]` , for example:
 - `char name[12];` // name with 11 characters
 - `char Address[50];` // address with 49 characters
- With initialization
 - `char identifier=string constant` , for example:
 - `char name[5]="John";` //name with 5 characters
 - `char name[]="John";` //also ok
 - `char name[5]={'J','o','h','n','\0'};` //also ok
- **However, you cannot initialize a string after declaration**
 - `char name[10];`
 - `name="john";` // wrong!

Compilation error : '=' : cannot convert from
'const char [5]' to 'char [10]'

cstring : Reading and Printing

```
#include<iostream>
using namespace std;
void main() {
    char word[20];
    //read a string
    cin >> word;
    cout << word; //print a string
}
```

The array word can store 20 characters but we can only store up to 19 “useful” characters (need to reserved 1 char for ‘\0’)

Note: cin would not check the size of array, hence inputting long string (>19 chars) can result in runtime error!

Reading a line of characters

- `cin >> str` will stop when white space characters (*<space>*, *<tab>*, or *<enter>*) are encountered
- Suppose "hello world" is inputted

```
char s[20];  
cin >> s; //read ONLY "hello"  
cin >> s; //read "world"
```

- Therefore, `cin >>` could not be used directly if the string contains spaces

Read one line with `cin.getline`

- Member function of `cin` to read in a line of text
*(including spaces) up to the <enter> character
(or until the specified number of characters are read)*
- Two arguments:
 - char array to receive the input
 - size of the char array

```
#include <iostream>
using namespace std;
void main(){
    char s[20];
    cin.getline(s,20);
    cout << "\"" << s << "\"" << end;
}
```

Suppose "hello world" is inputted
This time you get the full string
but not just "hello"

The Null Character '\0'

- The null character '\0', is used to mark the end of a cstring
- '\0' is a single character (*although written in two symbols*)
- ASCII code of the null character is 0

H	e	l	l	o	\0				
c	s	2	3	1	1	\0			

Why we need '\0'?

- cstring is stored in main memory *continuously*
- When the cstring is passed to functions (e.g. output with *cout*), only the start address is passed. *(because the string is array)*.
- *cout* actually have NO IDEA how long the string is!
- *cout* simply keeps printing until '\0' is encountered.

What if there's no `\0`?

```
#include <iostream>
using namespace std;
void main() {
    int a=98765432;
    char s[]="Hello world";

    s[11]=' ';    //deliberately overwrite the '\0'
    cout << s <<endl;
}
```



```
C:\> C:\Program Files\Microsoft Visual Studio\VC\bin\cl.exe *.cpp
Hello world x
x?!!
Press any key to continue
```

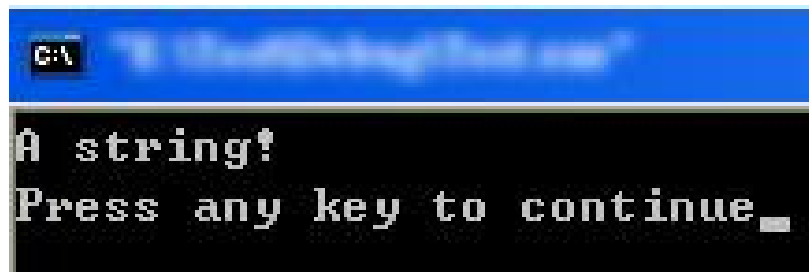
The role of '\0'

Similarly, we can artificially use '\0' to truncate a string

```
#include <iostream>
using namespace std;
void main(){
    char s[]="A string! A long long long long string!";

    s[9]='\0';    //cut at position 9
    //although characters 10...end are not changed,
    //but they are not printed as they are after '\0'

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



```
C:\> C:\Program Files\Microsoft Visual Studio\VC98\bin\cl.exe
A string!
Press any key to continue.
```

Passing strings to functions

- Example:
 - Write a `count` function to find the number of occurrences of a character in a string
 - Input:
 - `s`: The string to search from
 - `c`: The character to count

Note: As string is character array, it is passed by-reference! Be careful not to write on it otherwise the string in main will also be changed.

Function : count

The size 100 is optional, you may just write:
`int count (char s[], char c)`

```
int count(char s[100], char c){  
    int occurrence=0;  
    int i; //counter  
    for (i=0; s[i]!='\0'; i++){  
        if (s[i]==c)  
            occurrence++;  
    }  
    return occurrence;  
}
```

Should not write `i<100` here as the condition since the string could be much shorter!

Writing `i<100` will read 'garbage' after the end of string

The main function

```
void main() {  
    char str[50]="Hong Kong is a very  
    good place to live";  
    int count = count(str, 'o');  
    cout << "count = " << count << " \n");  
}
```



Similar to using array, it's WRONG to write:
count(str[], 'o'); or
count(str[50], 'o');

Another example: string length

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

int strlenh(char s[]){
    int i=0;
    while (s[i]!='\0')
        i++;
    return i;
}

void main(){
    char s[20]="Hello world";
    cout << strlenh(s);
}
```

Copy cstrings without using library

- Suppose there are two strings, `str1` and `str2`
- `str1` is initialized to "Hello world"
- Copy `str1` to `str2`
 - We **CANNOT** use the following expression to copy array elements

```
str2=str1;
```

- Similarly, we **CANNOT** use the following expression to compare strings

```
if (str1==str2) . . . . .
```

Copy cstrings without using library

- Use a loop to read characters one by one from `str1` until a null character (`'\0'`) is read
- Copy the character to the corresponding position of `str2`
- **IMPORTANT:**
Remember to put a null character at the end of `str2`

The program

```
#include <iostream>
using namespace std;
void main() {
    char str1[20]="Hello world";
    char str2[20];
    int i;

    /*beware of the boundary condition!*/
    for (i=0; i<19 && str1[i]!='\0'; i++) {
        str2[i]=str1[i];
    }
    str2[i]='\0'; //IMPORTANT
    cout << str2;
}
```

Exercise

- Write a program to let the user input a **line** of string
- The program should reverse the case of the input characters and print the result
 - Lowercase characters are changed to uppercase.
 - Uppercase characters are changed to lower case
- Example input/output:

```
Hello World  
hELLO wORLD
```

Program – fill in the missing parts

```
#include<iostream>
using namespace std;
void main(){
    char s[20];
    int i;
    cin.getline(s);

    for (i=0; s[i]!='\0'; i++){
        if (_____) //uppercase letter
            cout <<_____; //convert to lowercase
        else if (s[i]>='a' && s[i]<='z') //lowercase letter
            cout <<_____; //convert to uppercase
        else //other characters
            cout <<_____;
    }
}
```

Answer

```
#include<iostream>
using namespace std;
void main(){
    char s[20];
    int i;
    cin.getline(s);

    for (i=0; s[i]!='\0'; i++){
        if (s[i]>='A' && s[i]<='Z') //uppercase letter
            cout << s[i]+'a'-'A'; //convert to lowercase
        else if (s[i]>='a' && s[i]<='z') //lowercase letter
            cout << s[i]+'A'-'a'; //convert to uppercase
        else //other characters
            cout << s[i] ;
    }
}
```

Summary

- Character is a 8-bit integer (*corresponding to ASCII code*)
- Default `cin>>` of char skips `<space>`. You may use `cin.get(...)` instead
- CString is an array of character terminated by `'\0'`
(*i.e. char Name[20] can store only 19 characters for the name*)
- Access character from string is similar to accessing individual element of an array
- String cannot be copied (`=`) and compared (`==`) directly
- Default `cin>>` of string stops when `<space>` is encountered. You may use `cin.readline (...)` to read the whole line (*until <enter>*)
- String copy, comparison, concatenate can be done by functions provided by `<cstring>` library

For Reference Only

The <cstring> library

- Instead of writing loops on your own, you may actually use the cstring library for:
- String copy ([strcpy](#))
- String compare ([strcmp](#))
- Find the length of string ([strlen](#))
- Merging strings ([strcat](#))
- Find a character from string ([strchar](#))
- and more...

Some string functions in `<cstring>` library

Function	Description	Remarks
strcpy (dest,src)	Copy the content of string src to the string dest	No error check on whether the size of dest is enough for holding src
strcat (dest,src)	Append the content of string src onto the end of string dest	No error check on whether the size of dest is enough for holding src
strcmp(s1,s2)	Lexicographically compare two strings, s1 and s2, character by character.	0: s1 and s2 are identical >0: s1 is less than s2 <0: s1 is greater than s1
strlen(string)	Returns the number of characters (exclude the null character) in the string	

Examples:

```
#include <iostream>
#include <cstring>
using namespace std;
void main(){
    char s1[50]="String 1";
    char s2[50]="cs2363 computer programming";
    cout <<"s1:"<< s1 <<endl;
    cout <<"s2:" << s2<<endl;

    strcpy(s1,"cs2311");
    cout <<"s1:"<< s1 <<endl;

    strcat(s1," computer programming");
    cout <<"s1:"<< s1 <<endl;

    cout <<"Compare s1 vs s2: " << strcmp (s1,s2) <<endl;
    cout <<"Compare s2 vs s1: " << strcmp (s2,s1) <<endl;
    cout <<"Compare s1 vs \"cs2311 computer programming\": " <<
    strcmp (s1,"cs2311 computer programming") <<endl;
}
```

Example

- Write a program to print a word backward.
- Example:
 - Input:
 - hello
 - Output:
 - olleh

The program – fill in the missing parts

```
#include<iostream>
#include<cstring>
using namespace std;
void main() {
    char ____; //define an array input with size 20
    int n; //length of str
    int i;

    cin >> ____;
    n=____; //compute string length
    for (i=____; i >= 0 ;i--)
        cout << ____ ;
}
```

The program

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

void main(){
    char input[50]; //define an array input with size 50
    int n; //length of str
    int i;

    cin >> input;
    n=strlen(input); //compute string length

    for (i=n-1; i>=0; i--)
        cout << input[i];
}
```

cin.ignore (count, delimiter)

- Function to skip number of characters from the keyboard buffer *(or until the delimiter is encountered)*
- *count*: maximum number of characters to discard.
- *delimiter*: a character which signals the end of the *ignore/discard* operation. *(Note: the “delimiter” is also removed)*
- Example:
 - `cin.ignore(1000, '\n');`
/ skip at most 1000 characters or until '\n' is found, */*
 - `cin.ignore(1000, 'c');`
/ skip at most 1000 characters or until 'c' is found, */*

Example

```
#include <iostream>
using namespace std;
void main(){
    char name[20];
    char address[20];

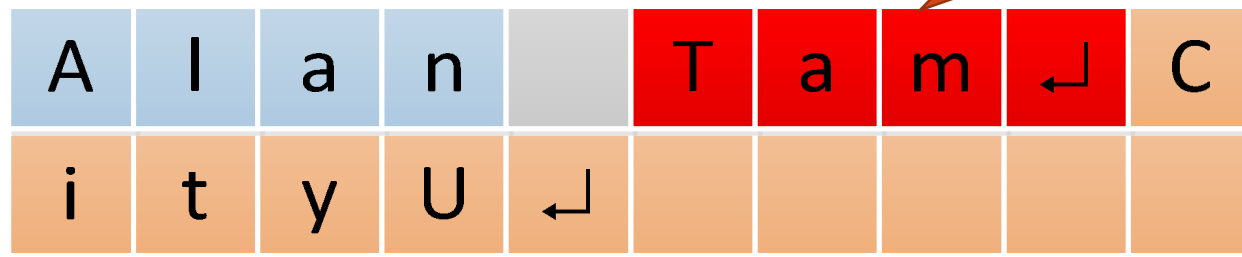
    cin >> name;
    cin.ignore(1000, '\n');
    cin >> address;
    cout << name << ':' << address << endl;
}
```

cin.ignore (count, delimiter)

```
C:\Windows\system32\cmd.exe  
Alan Tam  
CityU  
Alan:CityU  
Press any key to continue . . .
```

Store in cstring "name"

Discarded by
cin.ignore(..)



cin Error states

- From time to time, user's input could be deviated from the expectation
 - e.g. `cin>>integer` but user input string
 - e.g. `cin.getline` and user typed more than the array size
- For cases like that, `cin.fail()` will report `true`
- The “error” status persists and may hinder later `cin` operations
- Solution:
 - Call `cin.clear()` to cancel all “error” status
 - Possibly need to call `cin.ignore()` to remove incorrect input from the buffer

cin.fail() and cin.clear()

Let's observe the different behaviors when string / integer are typed

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

void main(){
    int x;
    while(1) {
        cin>>x;
        if (cin.fail()) {
            cout<<"Error, input is not integer!\n";
            cin.clear();
            cin.ignore(1000, '\n'); //remove all illegal chars until '\n'
        } else {
            cout<<"OK, input is integer "<<x<<endl;
        }
    }
}
```