CS2310 Computer Programming

LT04: Character, String and Structure

Computer Science, City University of Hong Kong (Dongguan)

Semester A 2025-26

Outline

- **Character**
  - Declaration and Initialization
  - Input and Output
- **String**
  - Declaration and Initialization
  - Input and output
  - Operations
- **Structure**
  - Declaration and Initialization
  - Operations

Character data type

- Written between single quotes. Examples
  `'A', 'b', '*'`
- In C++ language, a `char` type is **represented** by an integer
- Therefore, a character can also be **treated** as an integer

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

cout << (int) ‘7’; cout << (char) 97;

`char` features

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 `ch` is a lower-case letter:

```
if ('a' <= ch && ch  <= 'z') ...
```

same as:

```
if (97 <= ch && ch <= 122) ...
```

If the variable `ch` is a lowercase letter, then the expression

```
ch - 'a' + 'A' /*same as ch -97 + 65*/
```

has the value of the corresponding uppercase letter

Code: lec04-01-char.cpp

Reading a character

```
char c1,c2;
cin >> c1;
```

When `cin >> c1` is reached, the program will ask the user for input.

Suppose the character 'A' is input, c1 will evaluate to 65 (which is the ASCII code of 'A').

As a result, 65 will be assigned to the character c1. Therefore, c1 holds the character 'A'

Some facts about keyboard Input

- Suppose **>>** is called to read a character.
- What if the user input more than 1 characters in a single line?
- Answer: The extra character will be stored in a buffer (certain memory location). The character will be retrieved later when >> is called to read more characters

Some facts about keyboard Input

```
char c1,c2,c3;
cin >> c1; //enter the string "CS2310"
cin >> c2; //get the character 'S' from buffer
cin >> c3; //get the character '2' from buffer
```

	c1	c2	c3	Input buffer
(user input " cs2310")				C S 2 3 1 0
cin >> c1;	'C'			S 2 3 1 0
cin >> c2;	'C'	'S'		2 3 1 0
cin >> c3;	'C'	'S'	'2'	3 1 0

Printing a character

```
char c1='A',c2='B';
cout << c1;
cout.put(c1);
```

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.)

Answer

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

Answer

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

Outline

- **Character**
  - Declaration and Initialization
  - Input and Output
- **String**
  - Declaration and Initialization
  - Input and output
  - Operations
- **Structure**
  - Declaration and Initialization
  - Operations

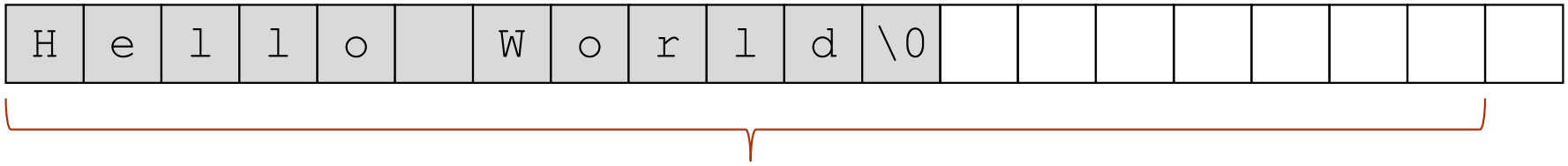
cstring vs string object

- In C++, there are two types of strings
- cstring: inherited from the **C** language
- string: class defined in <string> **library**

String : cstring

- A cstring is a `char` array terminated by `'\0'` (which is named **null character**) representing the end-of-string sentinel
- A character array of size **`n`** may store a string with 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**

cstring: Declaration and Initialization

- String variable can be declared in one of two ways
- Without **initialization** (with one more character than needed)
  - char identifier[required size+1];
  - E.g.
    - char **name**[12]; // **name** with 11 characters
    - char **Address**[50]; // **Address** with 49 characters
- With **initialization**
  - char identifier[]=string constant
  - E.g.
    - char name[]="John"; //name with characters
    - char choice[]="a,b,c,d,e"; //choice with characters
- However, you cannot initialize a string after declaration
  - char name[10];
  - name="john";

error C2440: '=' : cannot convert from 'const char [5]' to 'char [10]'

Code: lec04-02-stringdef.cpp

Example

```
void main
{
                    int mark;
                    char grade;

                    cin>>mark;

                    if(mark>80)
                              grade="A";
                    else if(mark>60)
                              grade="B";
                    else if(mark>50)
                              grade="C";
                    else
                              grade="F";

                    cout<<grade<<
}
```

error C2440: '=' : cannot convert from 'const char [2]' to 'char'

cstring : Reading and Printing

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

    cout << word; //print a string
}
```

The array word can store 20 characters but we can only we up to 19 character (the last character is reserved for null character).

Reading a line of characters

- `cin >> str` will **terminate** when whitespace characters (space, tab, linefeed, carriage-return, formfeed, vertical-tab and newline characters) is encountered

- Suppose "hello world" is input

```
char s1[20],s2[10];
cin >> s1; //user input "hello world"
cin >> s2; //does not require user input
cout << s1; //output "hello"
cout << s2; // output "world"
```

- How to read a line of characters (before '\n' is encountered)?

Code: lec04-03-stringread.cpp

cin.get

- get(): member function of cin to read in one character from input
- Syntax:

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

How to design?

1. cin.get + while loop

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

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


Code: lec04-04-cinget.cpp

2. cin.getline

- Predefined member function of cin to read a line of text (including space)
- Two arguments:
  - cstring variable to receive the input
  - maximum number of characters to get (including '\0')

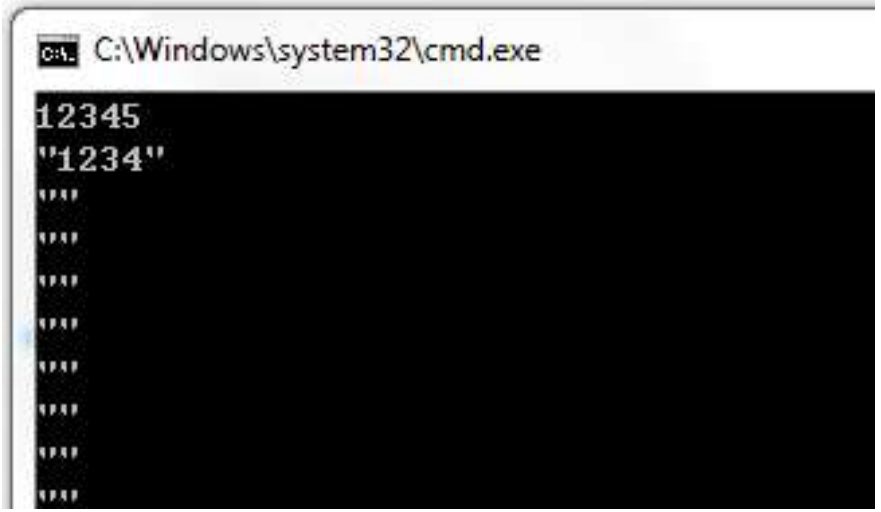
```
#include <iostream>
using namespace std;
void main(){
        char s[20];
        while (1){
                cin.getline(s,20);
                cout << "\"" << s << "\"" << endl;
        }
}
```


Code: lec04-05-getline.cpp

Example

```
#include <iostream>
using namespace std;
void main(){
        char s[5];
        while (1){
                cin.getline(s,5);

                cout << "\"" << s << "\"" << endl;
        }
}
```



cin.getline

- What if
  - Input is longer than the string variable?
  - **End** of the source characters is reached?
  - **Error** occurred?
- Internal state **flags** (eofbit,failbit,badbit) of cin object will be set
- To reset those flags, call method **clear()** of cin. E.g. **cin.clear()**;

Example

```
#include <iostream>
using namespace std;
void main(){
        char s[5];
        while (1){
                cin.getline(s,5);
                cin.clear();
                cout << "\"" << s << "\"" << endl;
        }
}
```

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 => escape sequence)
- It is used to distinguish a cstring variable from an ordinary array of characters (cstring variable must contain the null character)

Why need '\0'?

- cstring is stored in main memory continuously
- **Only** the starting address of the cstring is stored in cstring variable

```
char s1[]="Hello World";      // s1=20
char s2[]="cs2310";           // s2=32
```

- '\0' indicates the **end** of cstring

	0	1	2	3	4	5	6	7	8	9
0	r	o	g	r	a	m	m	i	n	a
1	O	-	m	a	4	1	.	;	t	a
2	H	e	l	l	o		w	o	r	l
3	d	\0	c	s	2	3	1	0	\0	&
4	1	*	~	^	b	/	a	v	e	

Why need '\0'?

- When a cstring variable is passed to an output function, i.e. cout, the function will print all the memory content until '\0' is encountered

	0	1	2	3	4	5	6	7	8	9
0	r	o	g	r	a	m	m	i	n	a
1	O	-	m	a	4	1	.	;	t	a
2	H	e	l	l	o		w	o	r	l
3	d	\0	c	s	2	3	1	0	\0	&
4	1	*	~	^	b	/	a	v	e	

Code: lec04-06-null.cpp

Why need '\0'?

```
#include <iostream>
using namespace std;
void main(){
        char s1[]="cs2310";
        char s2[]="Hello world";

        s2[11]=' ';//change '\0' to a space character ' ';
        cout << s2;
        cout << endl;
}
```

Passing strings to functions

- Example:
  - Write a function to count the frequency of a character (e.g., ‘a’) in a string
- Functions
  - `count`: given a character and a string as input, return the frequency of the character in the string
  - `main` function: call `count` function

Function : count

The size **100** is optional

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

Function : count

The size **100** is optional

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

The main function

```
void main(){
  char str[50]=”CityU is a very good
  university";
  int freq = count(str, 'a');
  cout << " freq = “ << freq << “ \n");
}

int count(char s[100], char c)
{
  …
}
```

Outline

- **Character**
  - Declaration and Initialization
  - Input and Output
- **String**
  - Declaration and Initialization
  - Input and output
  - Operations
- **Structure**
  - Declaration and Initialization
  - Operations

Common cstring functions in <cstring>

Function	Description	Remarks
strcpy (**dest**, **src**)	Copy the content of string **src** to the string **dest**	No error check on 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 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** is identical >0: **s1** is greater than **s2** <0: **s1** is less than **s1**
strlen(**str**)	Returns the number of characters (exclude the **null character**) contain in string **str**	

Note: you may need to use **strcpy_s** and **strcat_s** instead of **strcpy** and **strcat** if you are using the latest visual studio.

strcpy (dest, src), strcat (dest, src)

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

int main() {
  char src[] = "This is CS2310";
  char dest[40];
  strcpy(dest, src);
  cout << dest << endl;
  strcat(dest, " Lecture.");
  cout << dest << endl;
  return 0;
}
```


Code: lec04-07-copy.cpp

strlen(str)

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

int main() {
  char str[50];
  long len;
  strcpy(str, "This is CS2310");
  len = strlen(str);
  cout << "The length of str is " << len <<
    endl;
}
```


Code: lec04-08-len.cpp

Try to implement: string length

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

int length_of_string (char s[]){
  int length = 0;
  while (s[length]!='\0')
      length++;
  return length;
}
void main(){
  char s[20]="Hello world";
  cout << length_of_string(s);
}
```

Example

- Write a function *mergeStrings()* with two strings as input, e.g., *str1* and *str2*
  - *str1*: BADG, *str2*: FHCE
- **Merge** str1 and str2 by an increasing order
  - E.g., ABCDEFGH

str1: ABDG, *str2*: CEFH

- **Key** idea:
  - Step 1: **sort** each string first
  - Step 2: **merge** two sorted strings
    - **Define** a counter for each string
    - If a char from one string is **printed**, the counter of that string is **increased** by 1

Code: lec04-09-merge.cpp

Outline

- **Character**
  - Declaration and Initialization
  - Input and Output
- **String**
  - Declaration and Initialization
  - Input and output
  - Operations
- **Structure**
  - Declaration and Initialization
  - Operations

C++ Structure (**struct**)

- **Group** several related variables under one name
  - **Each variable** is a member of the **structure**
  - Unlike an array, these variables can have different data types
- Create a structure

```
struct struct_name {
   datatype member_name1;
   datatype member_name2;
   // …
} [var_name]; // optional
```

- Three ways of specifying a structure variable

```
struct Course {
  int courseID;
  double score;
} c1;              // 1) first way to specify a variable

c1.courseID = 1000;
c1.score = 80.5;

// 2) second way to sepecify a variable
struct Course c2; // Course cs2204; also works
c2.courseID = 1111;

// 3) third way to specify a variable
Course c3 = {2222, 90};
```

- **Access** structure members
  - `variableName.memberName;`

Code: lec04-10-structure.cpp

Pointers to Structures

- A structure can be pointed to by a **pointer** of its own type
  - For such pointers, we need to use the arrow operator (->) to access the structure members

```
struct Course {
    int courseID;
    double score;
};

int main() {
 Course c1; // struct Course c1; also works
 Course *p; // struct Course *p; also works

 p = &c1;

 p->courseID = 1111; // use arrow operator (->) rather than dot operator (.)
  cout << c1.courseID << endl;
 return 0;
}
```

Structure Array

- Create a structure first and use this structure to create an array

```
struct Course {
  int courseID;
  double score;
};

struct Course courses[3] = {
  {1111, 70},
  {2222, 80},
  {3333, 90}
};

for (int i=0; i<3; i++) {
  cout << courses[i].courseID << ": " << courses[i].score << endl;
}
```

Nested Structures

- A structure can be a member of another structure

```
struct TA {
    int stuID;
};

struct Course {
    int courseID;
    double score;
    TA myTA;
};

int main() {
  Course c1;
  c1.courseID = 1111;
  c1.myTA.stuID = 123456;
  return 0;
}
```


Code: lec04-11-neststructure.cpp

Passing Structures to a Function

- Structure variable

```
void printCourse1(Course c) {
        cout << c.courseID << endl;
        c1.courseID = 2222;
}
```

- Pointer

```
void printCourse2(Course *p) {
        cout << p->courseID << endl;
        p->courseID = 2222;
}

void main() {
  Course c;
  printCourse1(c);
  printCourse2(&c);
}
```

Critical Thinking

- Pointer with const
  - Preferred if the function does not suppose to modify the structure

```
void printCourse3(const Course *p) {
        // cannot modify the members of the structure pointed by p
}

void main() {
  Course c;
  printCourse3(&c);
}
```


Code: lec04-12-function.cpp