ETH zürich

Session 07
Strings, Recursion I

Informatik I

Lan Zhang

07.04.2025

Schedule

1. Exercise Feedback
2. Theory Recap
   - Chars & Strings
   - Recursion
3. In-Class Code Examples

Char

Represents printable characters and control characters `(\n, \t)`

- `char c = 'a'; // single quotation marks`
- Stored as 1 Byte (8 Bits) uint with value in range [0,128)
- Variables of type char can be treated as numbers

ASCII (American Standard Code for Information Interchange)

Defines conversion rules char <-> int

Decimal - Binary - Octal - Hex – ASCII Conversion Chart

'a'-'A' = 32
char to int: 'num'-'0'
encryption: (char-'a'+shift)%26 + 'a'

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

String

Corresponds to vector of char elements.

- Requires `#include<string>`
- `std::string s = “a” // double quotation marks`
- `std::string text(n,'a’) // Initializes string of length n filled with a`
- Strings can be compared `(text1 == text2)` and combined `(text1 += text2)`
- To access individual characters, use the same notation as for vectors. `string.at()/string[]`

String

Member functions:

- `s.size() / s.length()`
- `s1.append(s2, pos, n)` - appends n chars from s2 to s1, beginning at pos
- `s.substr(pos, n)` - returns substring from pos with n chars
- `s1.find(s2, (pos))`- (int) position of first occurrence of s from pos or npos if string is not found
- `s1.insert(pos, s2)` - inserts s2 at position pos in s1
- `s1.swap(s2)` - swaps s1 with s2

Recursion

Function calls itself during execution

“Divide and conquer”

- Split problem into smaller, easier subproblems
- Solve each subproblem separately
- Combine the small solutions to solve the full problem

```
unsigned int factorial(unsigned int n){
    //BASECASES
    if(n<2)return 1;
    //RECURSION
    return n*factorial(n-1);
}
```

Implementation

1. Base case
2. Work towards base case
3. Recursive call

Example
Compute n!

0! = 1! = 1

reduce n every recursive call

n! = n * (n-1)!

Recursion

Base Case

- Smallest possible problem
- Termination condition that is guaranteed to be reached
- Otherwise infinite recursion! consumes time and memory (stack overflow)

Call stack

- Each function call is pushed onto the call stack (First-In Last-Out queue)
- As soon as the base case is reached, the stack is processed from top to bottom

Recursion can be converted to iteration (loop) -> but not always easily or efficiently

In-Class Code Example

Upper Case

```
void char_to_upper(char& letter) {
     if ('a' <= letter && letter <= 'z') {
             letter -= 'a' - 'A';  // 'a' > ‘A’
     }
}
```

In-Class Code Example

Power function

```
void power(int x, int n){
  if(n==0) return 1;
  if(n<0){
     n = -n;
     x = 1/x;
  }
  if(n%2==0) return power(x*x,n/2);
  else return x*pow(x*x, n/2);
 }
}
```

In-Class Code Example

Partial Sum

```
unsigned int partial_sum(const unsigned int n) {
  if (n == 0) return 0;
  else {
    std::cout << n << std::endl; // print descending
    unsigned int partial = partial_sum(n - 1);
    std::cout << n << std::endl; // print ascending
    return n + partial;
  }
}
```

Tail Recursion: Recursive call is made after any other operations in the function.

Head Recursion: Recursive call is made before any other operations in the function. Each recursive call must wait for the subsequent calls to complete before it can proceed.

Recursion: Examples

```
void binary(int n){
  if(n==0) return;
  else{
    binary(n/2);
    cout << n%2;
  }
}
void log_2(int n){
  if(n==1) return 0;
  else{
    return 1 + log_2(n/2);
  }
}
```