CS1010E Lecture #10

Characters & Strings

NUS National University of Singapore
Department of Computer Science
School of Computing

Quiz

(CS1101C AY2006/07 Semester 2 Exam, Q10)

- What is printed out by the following C program?

```
// ...
int main(void) {
  int x[4][4] = {{ 1,  2,  3,  4}, { 5,  6,  7,  8},
                 { 9, 10, 11, 12}, {13, 14, 15, 16}};
  printf("%d\n", f(x));
  return 0;
}

int f(int a[4][4]) {
  int i, j, k = 0;
  for (i = 1; i < 4; i++) {
    for (j = 0; j < i; j++) {
      k += a[i][j];
    }
  }
  return k;
}
```

	[0]	[1]	[2]	[3]
[0]	1	2	3	4
[1]	5	6	7	8
[2]	9	10	11	12
[3]	13	14	15	16

Quiz

- What is printed out by the following program fragment?

```
int i, j, count = 0, num[10][10], newnum[100];

for (i = 0; i < 10; i++) {
  for (j = 0; j < 10; j++) {
    num[i][j] = i*10 + j;
  }
}

for (i = 0; i < 10; i++) {
  for (j = 0; j < 10; j++) {
    newnum[count] = num[i][j];
    count++;
  }
}

printf("%d %d\n", newnum[33], newnum[76]);
```

num	[0]	[1]	…	[9]
[0]	0	1	…	9
[1]	10	11	…	19
…	…	…	…	…
[9]	90	91	…	99

	[0]	[1]	[2]	…	[99]
newnum	0	1	2	…	99

Learning Objectives

- At the end of this lecture, you should understand:
  - How to declare and manipulate data of the char data type.
  - The fundamental operations on strings.

Characters

- In C, characters are represented using the data type **`char`**.
- Characters are written as symbols enclosed in single quotes.
  - Examples: `'G'`, `'8'`, `'*'`, `' '`, `'\n'`, `'\0'`

- Characters are stored in computer memory as integers using the ASCII scheme.
  - *ASCII* (American Standard Code for Information Interchange) is one of the document coding schemes widely used today.
  - *Unicode* is another commonly used standard for multi-language texts.

ASCII Table

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
10	lf	vt	ff	cr	so	si	dle	dc1	dc2	dc3
20	cd4	nak	syn	etb	can	em	sub	esc	fs	gs
30	rs	us	sp	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	}	|	~	del		

For example, character 'O' is 79 (row value 70 + col value 9 = 79).

Demo #1 : Using Characters

```c
#include <stdio.h>

int main(void) {

  char ch = 'a';

  printf("ch = %c\n", ch);
  printf("It's ASCII value = %d\n", ch);


  if ('A' < 'c') { // compare ASCII values
    printf("'A' is less than 'c'\n");
  } else {
    printf("'A' is not less than 'c'\n");
  }

  return 0;
}
```

declare a char variable

use %c to print out a character

use %d to print out ASCII value of a character

```
ch = a
It's ASCII value = 97
'A' is less than 'c'
```

Demo #2 : Character I/O

- Besides scanf() and printf(), we can also use getchar() and putchar() for character input and output.

```
#include <stdio.h>

int main(void) {

   char ch;

   printf("Enter a character: ");
   ch = getchar();  // scanf("%c", &ch);

   printf("Character entered is ");
   putchar(ch);     // printf("%c", ch);
   putchar('\n');

   return 0;
}
```

read a character from stdin

print a character to stdout

```
Enter a character: W
Character entered is W
```

Character Functions

Functions	Explanation
isalpha(c)	Returns a nonzero value if ***c*** is an English letter; return zero otherwise.
isupper(c)	Returns a nonzero value if ***c*** is an uppercase English letter; return zero otherwise.
islower(c)	Returns a nonzero value if *c* is a lowercase English letter; return zero otherwise.
isdigit(c)	Returns a nonzero value if ***c*** is a digit character ('0', '1', … '9'); return zero otherwise.
isalnum(c)	Returns a nonzero value if ***c*** is an English letter or a digit character; return zero otherwise.
isspace(c)	Returns a nonzero value if ***c*** is a blank, formfeed, newline, tab (i.e., whitespace); return zero otherwise.
tolower(c)	If ***c*** is an uppercase English letter, returns corresponding lowercase letter; returns ***c*** otherwise.
toupper(c)	If ***c*** is a lowercase English letter, returns corresponding uppercase letter; returns ***c*** otherwise.

Demo #3 : Character Functions

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

int main(void) {
  char ch;
  printf("Enter a character: ");
  ch = getchar();
  if ( isalpha(ch) ) { // is English letter?
    if ( isupper(ch) ) { // is upper case?
      printf("'%c' is a uppercase-letter.\n", ch);
      printf("Corresponding lowercase: %c\n", tolower(ch));
    }
  } else if ( isdigit(ch) ) { // is digit character?
    printf("'%c' is a digit character.\n", ch);
  } else if ( isspace(ch) ) { // is space?
    printf("'%c' is a whitespace character.\n", ch);
  }

  return 0;
}
```

to use character functions, need to include library <ctype.h>

tolower(ch) does NOT change ch itself!

```
Enter a character: B
'B' is a uppercase-letter.
Corresponding lowercase: b
```

Character Arrays

- An array in which all elements are characters.

```
#include <stdio.h>
#include <ctype.h>
int main(void) {

  int i;
  char fruit[5];

  for (i = 0; i < 5; i++) {
    scanf("%c", &fruit[i]);
  }
  for (i = 0; i < 5; i++) {
    putchar( toupper(fruit[i]) );
  }
  printf("\n");
  return 0;
}
```

fruit

a	p	p	l	e

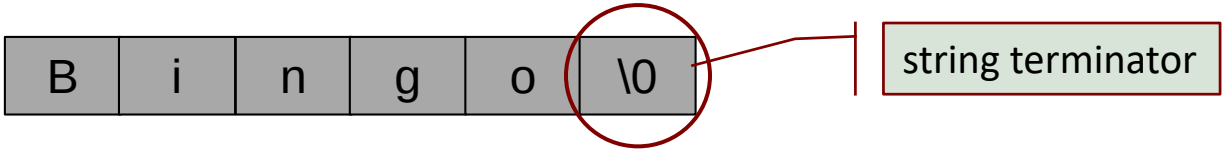
```
apple
APPLE
```

Strings

- String constant is commonly used in output statement.

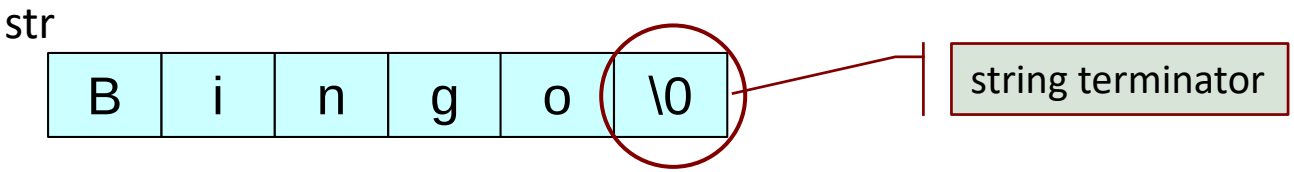
```
printf("Bingo");
```

  - In memory:



- A string is an array of characters, terminated by a null character '\0' (which has the ASCII value of zero).

```
char str[6] = {'B', 'i', 'n', 'g', 'o', '\0');
```



String Initializer

- Initializer for string
  - Using string constant, or char by char

string terminator auto added

a	p	p	l	e	\0

```
char fruit_name[] = "apple";
char fruit_name[6] = {'a', 'p', 'p', 'l', 'e', '\0'};
char fruit_name[6] = {'a', 'p', 'p', 'l', 'e'};
```

Q: Why 3rd line also gives a string?

- Some non-string examples:

```
char str[6];
```

?	?	?	?	?	?

```
char str[6];
str[0] = 'e';
str[1] = str[3] = 'g';
```

e	g	?	g	?	?

Strings : Output

```
char str[7] = "apple";
```

a	p	p	l	e	\0	\0

- Print string to screen

```
// str must be a string!
printf("%s\n", str);
```

```
apple
```

```
// str must be a string!
puts(str);                    // auto change line
```

- Output stops when the first '\0' is encountered.

Strings : Input a Word

- We use the following routine to read in a word (i.e. read up to and exclude a white space).

```
char name[5];
scanf("%s", name);          // read in a word
```

- Note the absence of the & operator in the second argument of **`scanf`**.

- Example: user input: `kiwi`

name

k	i	w	i	\0

Strings : Input a Sentence (1/2)

- We use the following routine to read in a line of input and store it as a string.

```
int len;
char str[8];

fgets(str, 8, stdin);  // read a line of input
len = strlen(str);     // find the length of string
if ( str[len-1] == '\n' ) {
  str[len-1] = '\0';
}
```

  - It will read up to 7 characters.
  - fgets will read in and store new line character '\n'.

user input: ya ya

str

y	a		y	a	\0	\0	?

Strings : Input a Sentence (2/2)

- Summary for fgets

```
fgets(str, size, stdin);
len = strlen(str);  // check length of string
if (str[len-1] == '\n') { // check the end of string
  str[len-1] = '\0';
}
```

  - Read at most size-1 characters, or till new line character.
  - New line char ('\n') might be read and stored in the string, therefore need to remove it if so.

There is another function gets(str) to read a string. However, we avoid using it due to security reason.

Demo #4 : Remove Vowels

- Write a program to remove all the vowels in a given string of at most 30 characters.

- Sample run:

```
Enter a string: Your name?
New string: Yr nm?
```

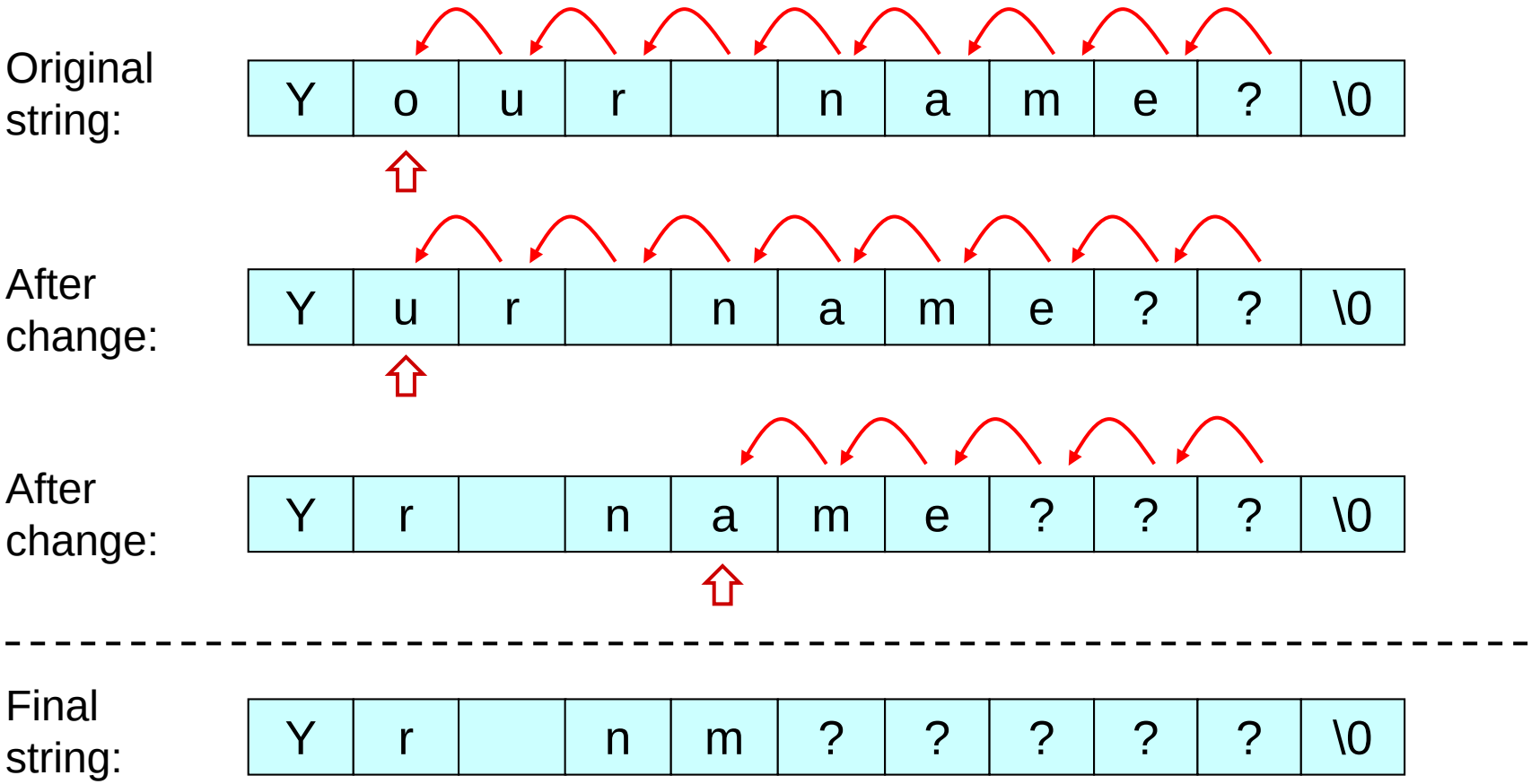
- Algorithm design: how can you figure out the answer manually by yourself?

Y	o	u	r		n	a	m	e	?	\0

Demo #4 : Algorithm 1

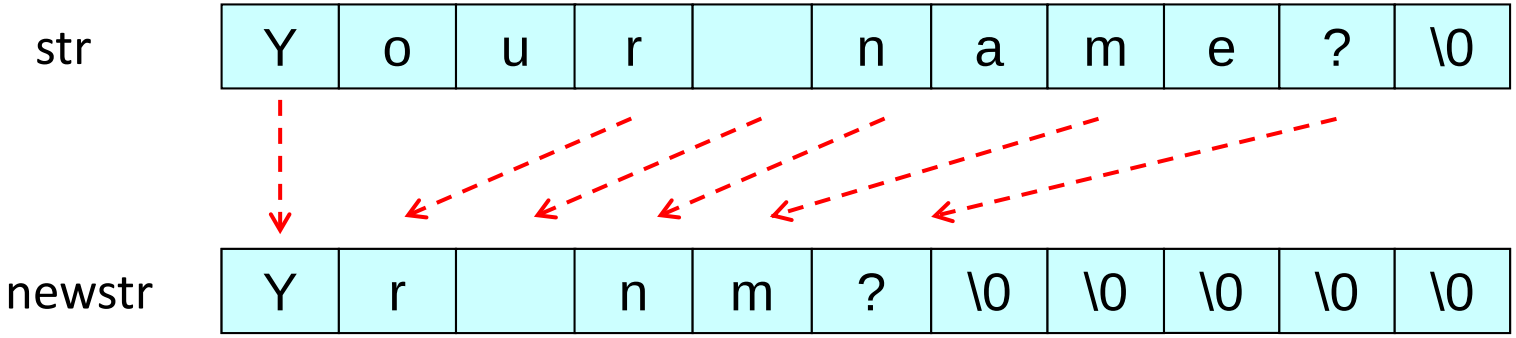
- Algorithm 1:

copy every element to the left by 1 position



Demo #4 : Algorithm 2

- Algorithm 2:



- A good algorithm save your life.
- Algorithms come from:
  - Book
  - Your experience (practice makes perfect)

We will implement algorithm 2

```c
#include <stdio.h>
#include <string.h> // to use strlen()
#include <ctype.h>  // to use toupper()

int main(void) {
  int i, len, index = 0;
  char str[31], newstr[31] = {'\0'};

  printf("Enter a string: ");
  fgets(str, 31, stdin);
  len = strlen(str);
  if (str[len-1] == '\n') { // clean up '\n' if any
    str[len-1]  = '\0';
  }

  len = strlen(str); // update string length
  for (i = 0; i < len; i++) {
    switch ( toupper(str[i]) ) {
      case 'A': case 'E': case 'I': case 'O': case 'U':
        break;  // ignore vowels
      default:  // copy non-vowel to newstr
        newstr[index] = str[i];
        index++;
    }
  } // end for loop
  printf("New string: %s\n", newstr);
  return 0;
}
```

Character Array w/o Terminator '\0'

- What is the output of the following code?

```
#include <stdio.h>
#include <string.h>
int main(void) {

  char str[10];
  str[0] = 'a';
  str[1] = 'p';
  str[2] = 'p';
  str[3] = 'l';
  str[4] = 'e';

  printf("Length = %d\n", strlen(str));
  printf("str = %s\n", str);

  return 0;
}
```

One possible output:

```
Length = 8
str = apple¿øp
```

a	p	p	l	e	?	?	?	?	?

A correct way is to add the following:

```
str[5] = '\0';
```

or write,

```
char str[10] = "apple";
```

%s and string functions work only on truly strings.

String Functions : Checking Length

- C provides a library of string functions
  - Must include <string.h>
  - http://www.cs.cf.ac.uk/Dave/C/node19.html
  - and other links you can find on the Internet

- **strlen(s)**
  - Return the number of chars in **s** before the first null character.

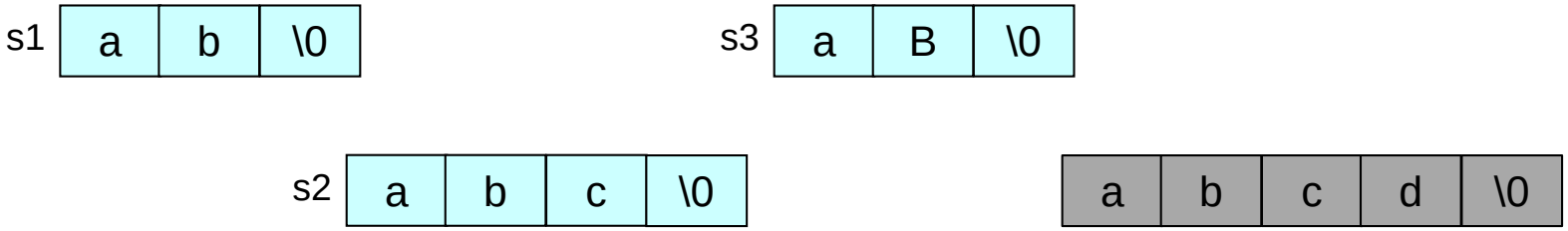
```
char s[12] = "Matthew Ho";
printf("%d", strlen(s));
```

s[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
M	a	t	t	h	e	w		H	o	\0	\0

String Functions : Comparison

- **`strcmp(s1, s2)`**
  - Compare **`s1`** and **`s2`** character by character, from left to right. Comparison stops once a difference is found.
  - Return
    - a negative integer if **`s1`** is less than **`s2`**, or
    - a positive integer if **`s1`** is greater than **`s2`**, or
    - 0 if **`s1`** and **`s2`** are equal

```
char s1[] = "ab", s2[] = "abc", s3[]= "aB";
printf("%d", strcmp(s1, s2) );
printf("%d", strcmp(s1, s3) );
printf("%d", strcmp(s3, "abcd") );
```



String Functions : Copying-and-Pasting

- **`strcpy(s1, s2)`**
  - Copy the string pointed to by **`s2`** into array pointed to by **`s1`**.

```
char name[10];
strcpy(name, "Matthew");
```

M	a	t	t	h	e	w	\0	?	?

- What if the string to be copied is too long?

```
strcpy(name, "A very long name");
```

A		v	e	r	y		l	o	n	g		n	a	m	e	\0

  - This is a logical error and program execution is unpredictable!
  - Moral: need to ensure destination string is big enough.

String Functions : Appending

- **`strcat(s1, s2)`**
  - Append a copy of string **`s2`**, including the terminating null character, to the end of string **`s1`**.

```
char s1[12] = "apple", s2[] = " pie";
strcat(s1, s2);
```

s1[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
a	p	p	l	e		p	i	e	\0	\0	\0

String Functions : Searching

- **`strchr(s, c)`**
  - Return a pointer to the first occurrence of character **`c`** in string **`s`**.
  - Return a NULL pointer if **`c`** is not found in **`s`**.

```
char *p = strchr("orange", 'a');
printf("%s\n", p);
```

```
ange
```

- **`strstr(s1, s2)`**
  - Return a pointer to the first occurrence of string **`s2`** in string **`s1`**.
  - Return a NULL pointer if **`s2`** is not found in **`s1`**.

```
char *p = strstr("orange", "ran");
printf("%s\n", p);
```

```
range
```

Demo #5 : String Functions

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

int main(void) {

  char s1[11] = "pineapples", s2[11] = "apple", *p;

  printf("strcmp(s1,s2) = %d\n", strcmp(s1, s2)); //comparison

  p = strstr(s1, s2);  // look for s2 in s1
  if (p != NULL) {
    printf("strstr(s1,s2) returns %s\n", p);
  } else {
    printf("strstr(s1,s2) returns NULL\n");
  }

  strcpy(s1,s2);  // copy s2 to s1
  printf("After strcpy, s1 = %s\n", s1);
  return 0;
}
```

```
strcmp(s1,s2) = 15
strstr(s1,s2) returns apples
After strcpy, s1 = apple
```

Today's Summary

Searching and Sorting

Characters

- ASCII values
- Character functions

Strings

- String initialization
- String input
- String functions