C CHARACTERS AND STRINGS

Deitel 8th Edition, Chapter 8

TOPICS

String Basics

Printing Strings

Reading Strings from stdin

String Library Functions

String Assignment

Substrings

Searching Strings

Tokenizing Strings

String Concatenation

Whole-Line Input

Comparing Strings

Lists of Strings

Working with Characters

String Summary

STRING BASICS

STRING BASICS – DEFINITIONS

String

- grouping of single characters
- characters can include numbers or punctuation

String constant

- One character, or a series of characters enclosed in **double quotes**
- Examples: “C” or “Programming in C”

Different than a **character constant**

- One character enclosed in **single quotes**
- Example: ‘C’

DECLARING A STRING VARIABLE

Same as declaring an array of type **char**

Examples:

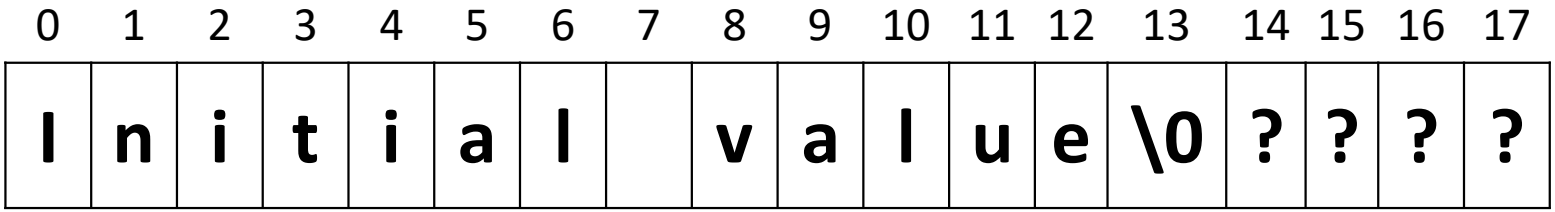
```
char  my_string[30];

char  first_name[15];
```

INITIALIZING A STRING VARIABLE

char student_name[18] = {“Initial value”};

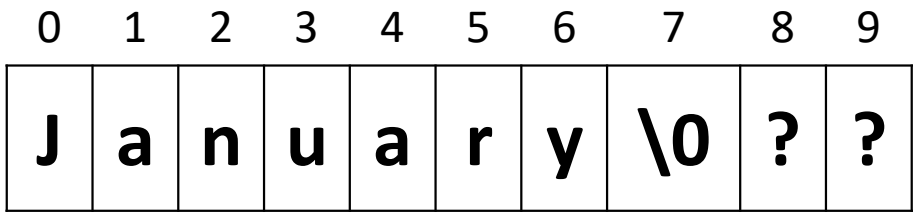
null character: \0

- indicates the end of a string
- automatically added for constant strings

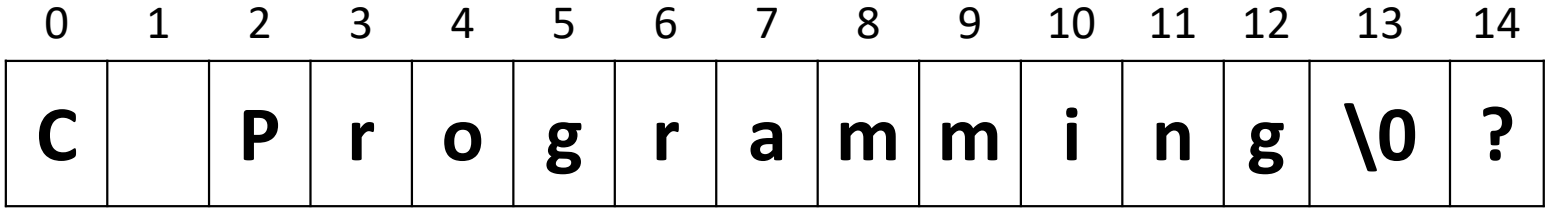


More Examples

char month[10] = “January”;



char title[15] = “C Programming”;



A String Name is Really a Pointer

Just like any array, the name of a character array is a constant pointer to the beginning of the array

So a string in C is accessed via a pointer to the first character in the string

We can also say that a string is a pointer—in fact, a pointer to the string's first character

A String Name is Really a Pointer 1

Example, using array syntax:

```
char color[] = "blue";
```

Can also be written using pointer syntax, as a pointer to a const char:

```
const char *colorPtr = "blue";
```

A String Name is Really a Pointer 2

```
char color[] = "blue";

const char *colorPtr = "blue";
```

Each initialize a variable to the string “blue”

First definition creates a 5-element array **color** containing the characters 'b', 'l', 'u', 'e' and '\0’

Second definition creates pointer variable **colorPtr** that points to the string "blue" somewhere in read-only memory

stringPointer.c

PRINTING STRINGS

PRINTING STRINGS – USE %S

Use **%s** as the format control string

printf will print until:

1. The **null character is reached**, or
2. A **run-time error occurs** because a memory cell was accessed that is not assigned to the program

USING PRINTF WITH STRINGS – EXAMPLE 1

```
char  dog_name[ 10 ] = “Scooby”;

char  cat_name[ 10 ] = “Grumpy”;

printf(“%s”, dog_name);

printf(“%s”, cat_name);
```

Output is:

USING PRINTF WITH STRINGS – EXAMPLE 1 – RESULTS

```
char  dog_name[ 10 ] = “Scooby”;

char  cat_name[ 10 ] = “Grumpy”;

printf(“%s”, dog_name);

printf(“%s”, cat_name);
```

Output is:

ScoobyGrumpy

To Right-Justify the Output, Use Field Widths

```
char  dog_name[ 10 ] = “Scooby”;

char  cat_name[ 10 ] = “Grumpy”;

printf(“%10s”, dog_name);

printf(“%10s”, cat_name);
```

Output is:

Scooby Grumpy

If field width too small, string is extended to the left.

To Left-Justify the Output, Add a Minus Sign

```
char  dog_name[ 10 ] = “Scooby”;

char  cat_name[ 10 ] = “Grumpy”;

printf(“%-10s”, dog_name);

printf(“%-10s”, cat_name);
```

Output is:

Scooby   Grumpy

How else can we print a string?

What is a string?

And how do we normally print those?

How do we know when to stop printing?

Where is that point in a string?

scooby.c

READING STRINGS FROM STDIN

USING SCANF WITH STRINGS

String variable in a scanf does not need the **AddressOf** operator

- Why not?

For each **%s** placeholder:

1. Starts scanning at **FIRST NON-WHITESPACE** character
2. Stops scanning at **NEXT WHITESPACE CHARACTER**
3. Automatically adds **NULL CHARACTER** at the end of each string

input_output.c

USING SCANF WITH STRINGS – EXAMPLES

```
char course_name[10];
int course_num;
```

Technique 1:

```
scanf("%s", course_name);
scanf("%d", &course_num);
```

Technique 2:

```
scanf("%s%d", course_name, &course_num);
```

STRING LIBRARY FUNCTIONS

String Library Functions Built into C

Many string manipulation functions built into C

String library functions contained in **string.h**

String Library Functions Are:

strcpy / strncpy

- String assignment
- Copies strings
- Extracts substrings

strlen

- Determines length of a string

strcat / strncat

- Concatenation

strcmp / strncmp

- Compares strings

String Assignment

STRING ASSIGNMENT – STRCPY 1

Cannot "assign" a value to a string (except at initialization)

- Remember, strings are arrays…

So we cannot write: courseName = "C Programming";

To assign a string a value, must use function **strcpy**

strcpy (destination string, source string);

strcpyExamples.c

STRING ASSIGNMENT – STRCPY 2

Destination string must be long enough to hold the source string, including a position for the null terminator

If destination is longer than source, destination is **null-filled**

If destination is shorter than source, destination is filled until its end. The rest of the source string may or may not be lost.

- No null terminator! ☹

STRCPY **EXAMPLE 1**

```
char name[18];

strcpy(name, "Test String");
```

Destination **longer** than source, so the entire source string is stored in destination, and the leftover slots in the destination are null-filled.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
T	**e**	**s**	**t**		**S**	**t**	**r**	**i**	**n**	**g**	**\0**	**\0**	**\0**	**\0**	**\0**	**\0**	**\0**

STRCPY **EXAMPLE 2**

char name[18];

strcpy(name, "A very long test string");

Destination **shorter** than source, so the **t, r, i, n, g** and **\0** are stored in memory allocated for other variables.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
A		**v**	**e**	**r**	**y**		**l**	**o**	**n**	**g**		**t**	**e**	**s**	**t**		**s**

STRING ASSIGNMENT – STRNCPY 1

Can also give a string a value by indicating **how many characters of the source string** to use:

strncpy (destination string, source string, number);

strncpyExamples.c

STRING ASSIGNMENT – STRNCPY 2

Same rules as strcpy:

Destination string must be long enough to hold the source string, including a position for the null terminator

If destination is longer than source, destination is **null-filled**

If destination is shorter than source, destination is filled until its end. The rest of the source string may or may not be lost.

- No null terminator! ☹

STRNCPY **EXAMPLE 1**

char name[18];

strncpy(name, “Test String”, 4);

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
T	**e**	**s**	**t**	**\0**	**\0**	**\0**	**\0**	**\0**	**\0**	**\0**	**\0**	**\0**	**\0**	**\0**	**\0**	**\0**	**\0**

STRNCPY **EXAMPLE 2**

char name[18];

strncpy(name, "A very long test string", 20);

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
A		**v**	**e**	**r**	**y**		**l**	**o**	**n**	**g**		**t**	**e**	**s**	**t**		**s**

What would happen if we tried to print this:

- Using **%s**?
- Using **%c**?

USING STRNCPY – BEST PRACTICE – EXPLICITLY ADD NULL TERMINATOR

Always explicitly add the null terminator at last position in strings when using strncpy:

General format:

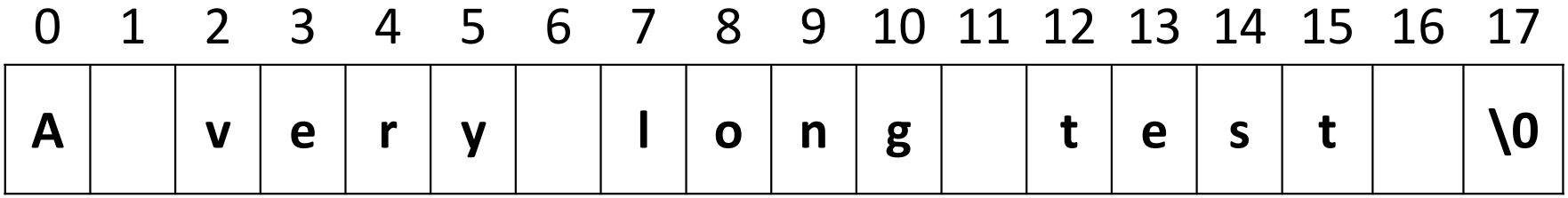
destination_string_name [length – 1] = '\0';

STRNCPY **EXAMPLE 3**

char name[18];

strncpy(name, “A very long test string”, 20);

name[17] = ‘\0’;



ANOTHER WAY TO GET A VALUE INTO A STRING – SPRINTF

"Prints" formatted data **into an array** instead of screen

Uses the same format control strings as **printf**

Fig08_12.c

SUBSTRINGS

SUBSTRINGS – DEFINITION

Substring – some part of a larger string

Can use **strncpy** to **extract substrings** from the middle of a larger string

```
strncpy( prefix, &course[0], 3);

strncpy ( number , &course[3], 3);

strncpy( &course[4], number, 3);
```

extractSubstringsExamples.c

USING STRNCPY TO EXTRACT SUBSTRINGS 1

Remember that an array name (without a subscript) represents the **address** of the array

Or more precisely, **the address of the first element** in the array

So the **source string of strncpy can start at any position in the array of chars**, if we tell it the address of this position.

USING STRNCPY TO EXTRACT SUBSTRINGS 2

In other words, we can start copying from the source string at the position of any character in the string.

How?

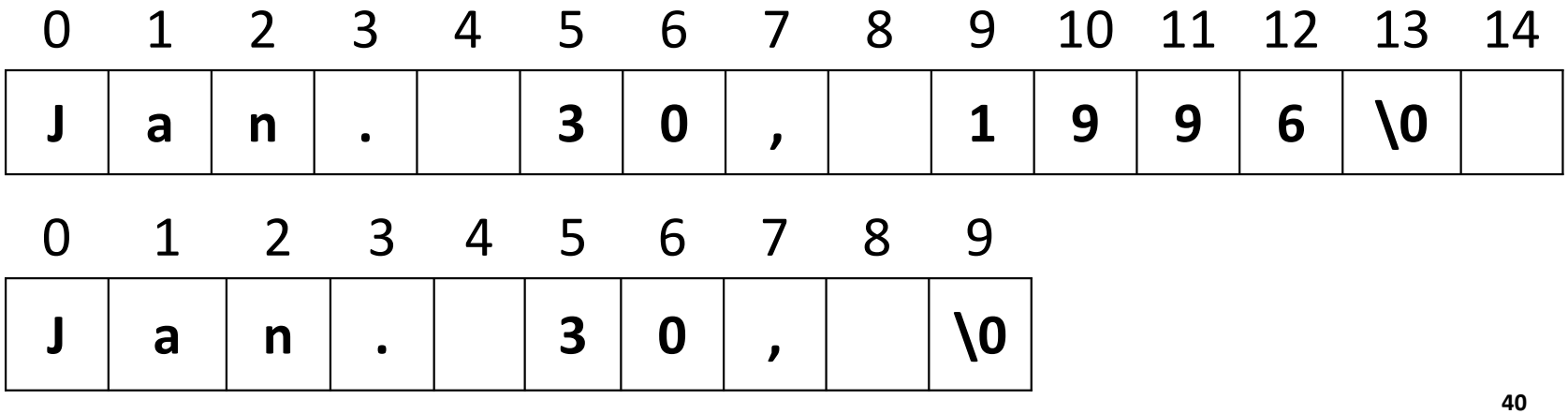
How do we write (in C) one of the characters in a string?

Or, how do we reference one element of our array of chars?

How do we indicate the address of that one element?

EXTRACTING SUBSTRINGS, EXAMPLE 1

```
char date[15] = "Jan. 30, 1996";

char result[10];

strncpy(result, date, 9);

result[9] = '\0';
```



EXTRACTING SUBSTRINGS, EXAMPLE 2

```
char date[15] = "Jan. 30, 1996";

char result[10];

strncpy(result, &date[5], 2);

result[9] = '\0';
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
J	a	n	.		3	0	,		1	9	9	6	\0	

0	1	2	3	4	5	6	7	8	9
3	0	\0							\0

EXTRACTING SUBSTRINGS, EXAMPLE 3

Use **strcpy** to copy from some starting point to the end:

char date[15] = "Jan. 30, 1996";

char result[10];

strcpy(result, &date[9]);

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
J	**a**	**n**	**.**		**3**	**0**	**,**		**1**	**9**	**9**	**6**	**\0**	

0	1	2	3	4	5	6	7	8	9
1	**9**	**9**	**6**	**\0**					**\0**

ANOTHER WAY TO READ FROM A STRING – SSCANF

"Reads" formatted data **from a string** instead of keyboard

Uses the same format control strings as **scanf**

Fig08_13.c

SEARCHING STRINGS

Searching Strings – Character or Another String

Can search a string for a single character

Can search a string for another string

SEARCHING A STRING FOR A CHARACTER – STRCHR

strchr searches for the first occurrence of a character in a string

If character is found, **strchr** returns a pointer to the character in the string

If character not found **strchr** returns a NULL pointer

Fig08_20.c

- Searches for the first occurrences of 'a' and 'z' in "This is a test"

SEARCHING A STRING FOR ANOTHER STRING – STRSTR

strstr searches for the first occurrence of its second string argument in its first string argument

If the second string is found in the first string, a pointer to the location of the string in the first argument is returned

If the second string not found **strstr** returns a NULL pointer

Fig08_25.c

- Searches for the string "def" in the string "abcdefabcdef"

TOKENIZING STRINGS

TOKENS

A **token** is a sequence of characters separated by **delimiters**

- Usually spaces or punctuation marks, but can be any character

For example, in a line of text, each word can be considered a token, and the spaces and punctuation separating the words can be considered delimiters

To "**tokenize**" means to split up a longer sequence of characters into its separate pieces

- Think of the "words" in a "sentence"

TOKENIZING STRINGS – STRTOK 1

strtok is used to break a string into a series of tokens

Uses two arguments:

- The address of the string to be tokenized
- A string containing characters that separate the tokens (the delimiters)

Returns a pointer to the first token in the string

If there are no tokens, it returns NULL

TOKENIZING STRINGS – STRTOK 2

Fig08_26.c

Line 15: **char * tokenPtr = strtok(string, " ");**

- assigns tokenPtr a pointer to the first token in string
- Second argument, " ", indicates that tokens are separated by spaces

strtok searches for the first character in the string that's **not** a delimiting character

- The beginning of the current token

Then finds the **next delimiting character** in the string and replaces it with a null ('\0') character

- The end of the current token.

TOKENIZING STRINGS – STRTOK 3

Saves a pointer to the next character following the token in string and returns a pointer to the current token

Subsequent **strtok** calls use NULL as their first argument

- Indicates that the call to **strtok** should continue tokenizing from the location in string saved by the last call to **strtok**

String Concatenation

CONCATENATION

Concatenation means join

To **concatenate** two strings means to **put one at the end of the other**

So one string is changed, and the other is not

Use special string functions that modify the first string by adding all or part of the second string to the end of it

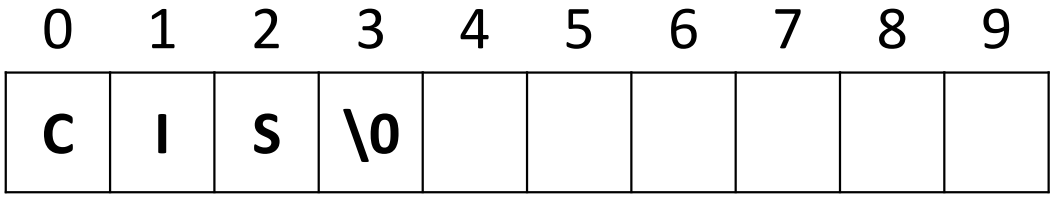
concatenationExamples.c

STRCAT **FUNCTION FOR CONCATENATION**

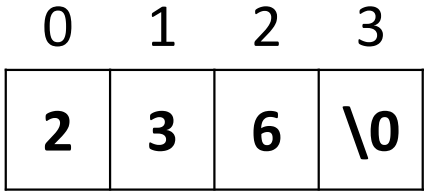
General format: **strcat (destination, source);**

Examples:

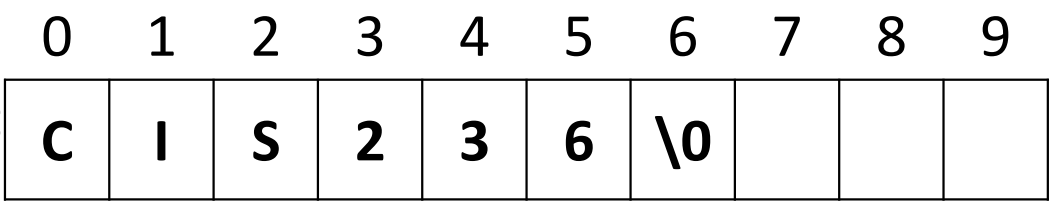
char course_name[10] = “CIS”;



char course_code[4] = “236”;



strcat(course_name, course_code);



STRNCAT **FUNCTION FOR CONCATENATION**

General format: **strncat (destination, source, number);**

Example:

char course_name[10] = “CIS”;

0	1	2	3	4	5	6	7	8	9
C	**I**	**S**	**\0**						

char course_code[4] = “236”;

0	1	2	3
2	**3**	**6**	**\0**

strncat(course_name, course_code, 2);

0	1	2	3	4	5	6	7	8	9
C	**I**	**S**	**2**	**3**	**\0**				

USING STRCAT & STRNCAT

Must make sure the total length of the two strings is **less** than the length of the destination

What would happen if there isn't "enough room"?

Use the **strlen** function to check the length of each string before you use strcat/strncat.

FINDING STRING LENGTH – STRLEN

Calculates length of string, **NOT including null character**, regardless of declared length of array

Example:

```
char name[10] = "Jack";

int length;

length = strlen(name);
```

The value of **length** after these statements execute is 4.

REMEMBER!

Remember these important questions when manipulating strings:

Is there **enough space in the created string** for the entire new string?

Does the created string end with **null character** '\0'?

Whole-Line Input

Whole-Line Input When Spaces are Valid

Remember that **scanf** stops at a **whitespace** character when reading strings

But spaces may be a valid part of a string (e.g. blank spaces within a file name or a sentence)

WHOLE-LINE INPUT – GETS FUNCTION

Use **gets** function to get a string from stdin (keyboard)

Stops at the newline **\n** (not first whitespace char like scanf)

Newline character NOT stored in string

WHOLE-LINE INPUT – GETS FUNCTION 2

General format: **gets (destination_string);**

Example:

```
char line[80];
printf("Type in a line of data:\n");
gets(line);
```

gets_example.c

COMPARING STRINGS

STRING COMPARISON

Previously, we've used relational and equality operators to compare **characters**:

'a' == 'A'

char1 < char2

Can't use these for strings, because strings are arrays, and we can't compare arrays directly

More special string functions!

STRING COMPARISON – STRCMP FUNCTION 1

Compares two strings, string1 and string2

General format:

return_value = strcmp (string1, string2);

STRING COMPARISON – STRCMP FUNCTION 2

strcmp returns an integer value

positive number if string1 > string2 alphabetically

- string1 appears **after** string2 in dictionary (on a page with a higher number)

zero if the strings are equal

negative number if string1 < string2 alphabetically

- string1 appears **before** string2 in dictionary (on a page with a lower number)

strcmpExamples.c

STRCMP **EXAMPLE 1**

```
char string1[20] = “cat”;

char string2[20] = “hamster”;

int return_value;


return_value = strcmp(string1, string2);
```

return_value is

STRCMP **EXAMPLE 1 – RESULTS**

char string1[20] = “cat”;

char string2[20] = “hamster”;

int return_value;

return_value = strcmp(string1, string2);

return_value is negative

STRCMP **EXAMPLE 2**

```
char string1[20] = “hedgehog”;

char string2[20] = “dog”;

int return_value;


return_value = strcmp(string1, string2);
```

return_value is

STRCMP EXAMPLE 2 – RESULTS

```
char string1[20] = "hedgehog";

char string2[20] = "dog";

int return_value;


return_value = strcmp(string1, string2);
```

return_value is positive

STRCMP **EXAMPLE 3**

```
char string1[20] = “fish”;

char string2[20] = “fishes”;

int return_value;


return_value = strcmp(string1, string2);
```

return_value is

STRCMP **EXAMPLE 3 – RESULTS**

char string1[20] = “fish”;

char string2[20] = “fishes”;

int return_value;

return_value = strcmp(string1, string2);

return_value is negative

STRING COMPARISON – STRNCMP FUNCTION

Compares two strings, string1 and string2, for some number of positions

General format:

return_value = strncmp (string1, string2, number);

strncmp return value is the same as strcmp:

- **positive** number if string1 > string2 alphabetically
- **zero** if the strings are equal
- **negative** number if string1 < string2 alphabetically

STRNCMP **EXAMPLE 1**

```
char string1[20] = “horse”;

char string2[20] = “bobcat”;

int return_value;


return_value = strncmp(string1, string2, 5);
```

return_value is

STRNCMP **EXAMPLE 1 – RESULTS**

```
char string1[20] = "horse";

char string2[20] = "bobcat";

int return_value;


return_value = strncmp(string1, string2, 5);
```

return_value is positive

STRNCMP **EXAMPLE 2**

```
char string1[20] = “spider”;

char string2[20] = “spider monkey”;

int return_value;


return_value = strncmp(string1, string2, 6);
```

return_value is

STRNCMP **EXAMPLE 2 – RESULTS**

char string1[20] = “spider”;

char string2[20] = “spider monkey”;

int return_value;

return_value = strncmp(string1, string2, 10);

return_value is negative

STRNCMP **EXAMPLE 3**

```
char string1[20] = “spider”;

char string2[20] = “spider monkey”;

int return_value;


return_value = strncmp(string1, string2, 6);
```

return_value is

STRNCMP **EXAMPLE 3 – RESULTS**

```
char string1[20] = "spider";

char string2[20] = "spider monkey";

int return_value;


return_value = strncmp(string1, string2, 6);
```

return_value is 0

LISTS OF STRINGS

Lists of Strings Are Also Arrays

To create a list of strings, we need to know:

- **How many strings** to store
- **How long each string** can be

We'll need an **array of strings**

- But a string is already an array (of chars)
- So we need an array of arrays, which is ???

LISTS OF STRINGS – DECLARATION

General format:

```
char  array_of_strings  [NUM_STRINGS] [MAX_LENGTH];
```

List of Strings – Example 1 – Initializing an Array of Strings

Example – create a list of the month names

Step 1 – Determine the number of names needed

- 12

Step 2 – Determine the length of the longest month name

- September is 9 letters long
- So, we need 10 positions in the name of each month (to account for the null terminator)

List of Strings – Example 1 – Initializing an Array of Strings, 2

And since we know the values to store in the array, we can initialize it at declaration:

```
char month[12][10] = {  “January”, “February”, “March”, “April”,
                        “May”, “June”, “July”,  “August”, “September”,
                        “October”,  “November”, “December”   };
```

Array of Strings

	0	1	2	3	4	5	6	7	8	9
0	J	a	n	u	a	r	y	\0		
1	F	e	b	r	u	a	R	y	\0	
2	M	a	r	c	h	\0				
3	A	p	r	i	l	\0				
4	M	a	y	\0						
5	J	u	n	e	\0					
6	J	u	l	y	\0					
7	A	u	g	u	s	t	\0			
8	S	e	p	t	e	m	b	e	r	\0
9	O	c	t	o	b	e	r	\0		
10	N	o	v	e	m	b	e	r	\0	
11	D	e	c	e	m	b	e	r	\0	

Lists of Strings – Example 2 – List of Animal Names

A list of animal names, entered by the user

Step 1 – Determine the number of names needed

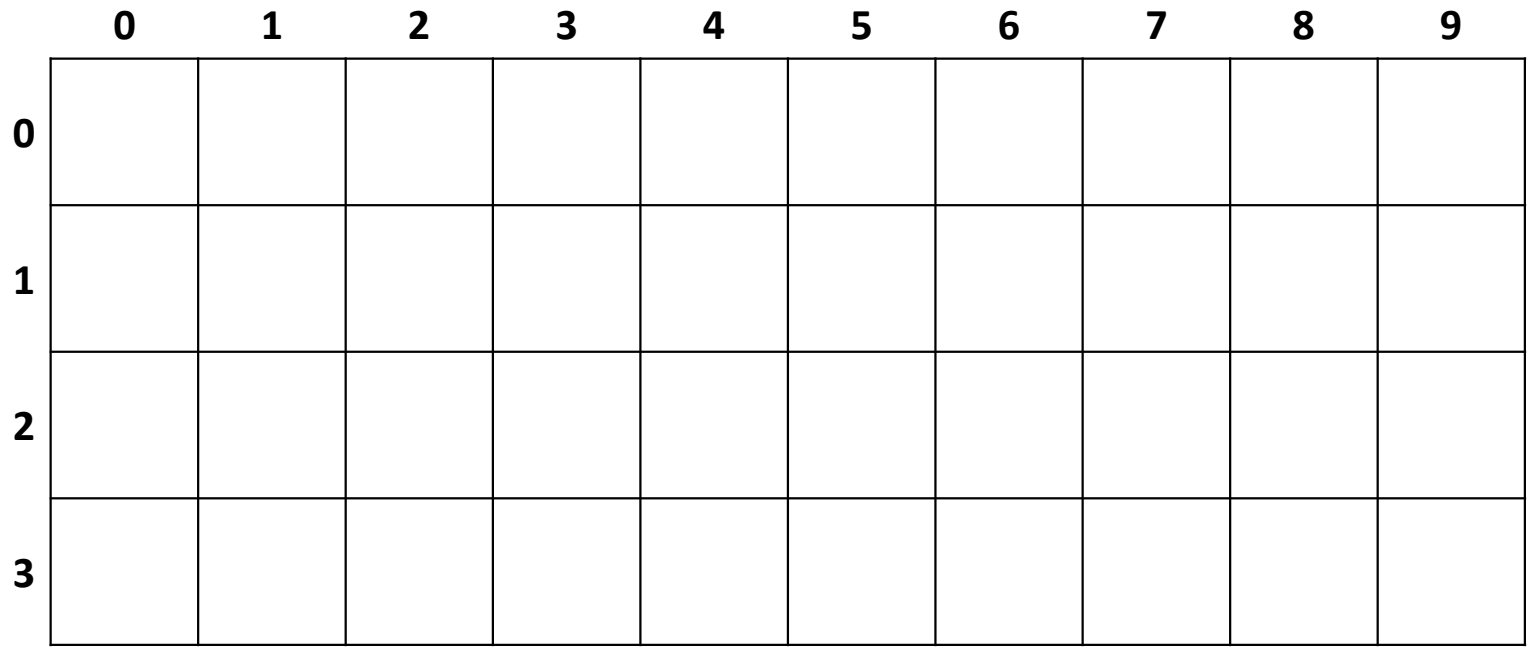
- For this example, we'll save 4 names
- And we'll use a symbolic constant for this value: **#define NUM_NAMES 4**

Step 2 – Decide on the maximum allowable length of each name (plus 1 for null!)

- For this example, we'll use 10
- And we'll use a symbolic constant for this value too: **#define MAX_LENGTH 10**

Lists of Strings – Example 2 – List of Animal Names 2

Then create the array: **char names [NUM_NAMES] [MAX_LENGTH];**



LISTS OF STRINGS – EXAMPLE 2 – LIST OF ANIMAL NAMES 3

Let's add some sample data to the **name** array:

	0	1	2	3	4	5	6	7	8	9
0	h	a	m	s	t	e	r	\0		
1	h	o	r	s	e	\0				
2	c	a	t	\0						
3	f	i	s	h	\0					

Lists of Strings – Example 2 – List of Animal Names 4

What is the **value** of names[0]?

	0	1	2	3	4	5	6	7	8	9
0	h	a	m	s	t	e	r	\0		
1	h	o	r	s	e	\0				
2	c	a	t	\0						
3	f	i	s	h	\0					

Lists of Strings – Example 2 – List of Animal Names 5

What is the **value** of names[0]? hamster

	0	1	2	3	4	5	6	7	8	9
0	h	a	m	s	t	e	r	\0		
1	h	o	r	s	e	\0				
2	c	a	t	\0						
3	f	i	s	h	\0					

Lists of Strings – Example 2 – List of Animal Names 6

But what **is** names[0]?

	0	1	2	3	4	5	6	7	8	9
0	h	a	m	s	t	e	r	\0		
1	h	o	r	s	e	\0				
2	c	a	t	\0						
3	f	i	s	h	\0					

Lists of Strings – Example 2 – List of Animal Names 7

But what **is** names[0]?

It's an array of chars, which is a string

	0	1	2	3	4	5	6	7	8	9
0	h	a	m	s	t	e	r	\0		
1	h	o	r	s	e	\0				
2	c	a	t	\0						
3	f	i	s	h	\0					

Lists of Strings – Example 2 – List of Animal Names 8

So each name in the list is a string, and to get the names into the array without using an initializer list, we need to use scanf with %s, or string functions

	0	1	2	3	4	5	6	7	8	9
0	h	a	m	s	t	e	r	\0		
1	h	o	r	s	e	\0				
2	c	a	t	\0						
3	f	i	s	h	\0					

STRING ARRAY ELEMENTS 1

What is the value of **names[0][2]**?

	0	1	2	3	4	5	6	7	8	9
0	h	a	m	s	t	e	r	\0		
1	h	o	r	s	e	\0				
2	c	a	t	\0						
3	f	i	s	h	\0					

String Array Elements 2

What is the value of **names[0][2]**? The single character m

	0	1	2	3	4	5	6	7	8	9
0	h	a	m	s	t	e	r	\0		
1	h	o	r	s	e	\0				
2	c	a	t	\0						
3	f	i	s	h	\0					

STRING ARRAY ELEMENTS 3

What is the value of **names[2][0]**?

	0	1	2	3	4	5	6	7	8	9
0	h	a	m	s	t	e	r	\0		
1	h	o	r	s	e	\0				
2	c	a	t	\0						
3	f	i	s	h	\0					

String Array Elements 4

What is the value of **names[2][0]**? The single character c

	0	1	2	3	4	5	6	7	8	9
0	h	a	m	s	t	e	r	\0		
1	h	o	r	s	e	\0				
2	c	a	t	\0						
3	f	i	s	h	\0					

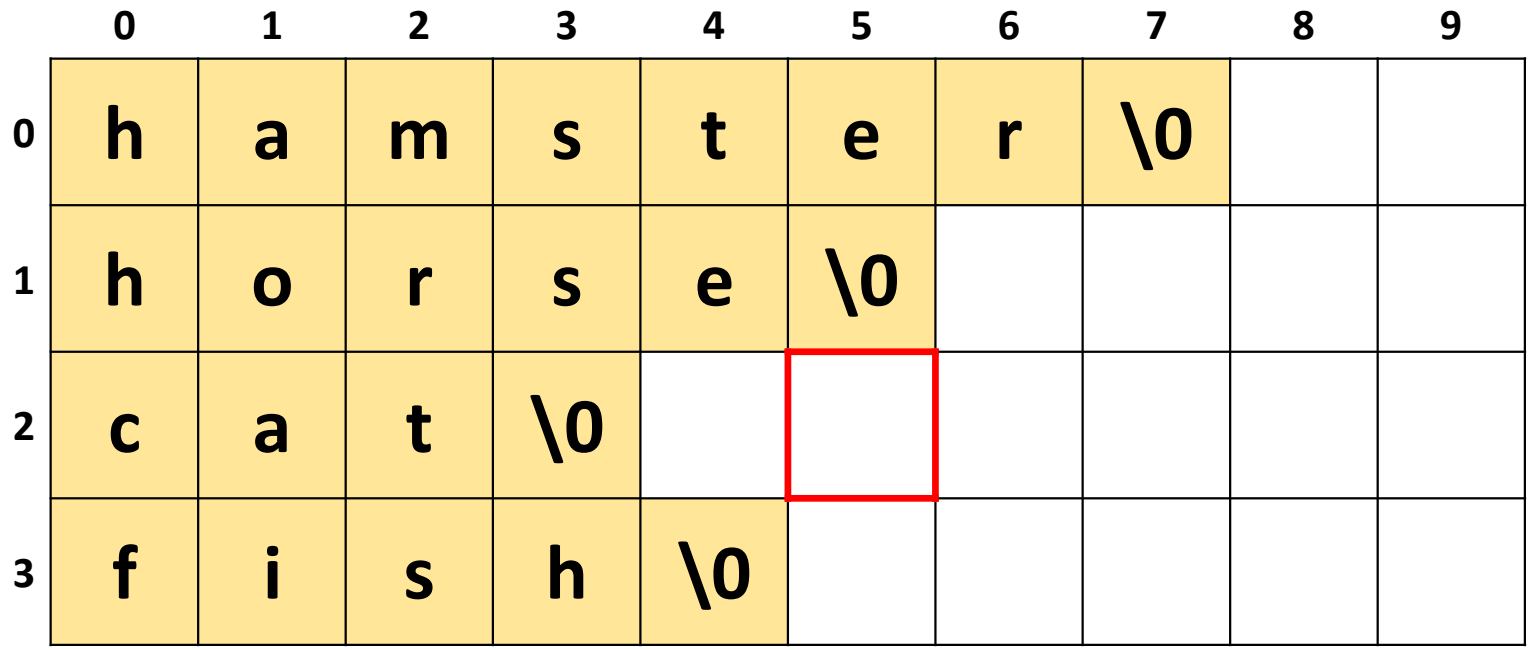
String Array Elements 6

What is the value of **names[2][5]**?

	0	1	2	3	4	5	6	7	8	9
0	h	a	m	s	t	e	r	\0		
1	h	o	r	s	e	\0				
2	c	a	t	\0						
3	f	i	s	h	\0					

String Array Elements 7

What is the value of **names[2][5]**? Unknown…



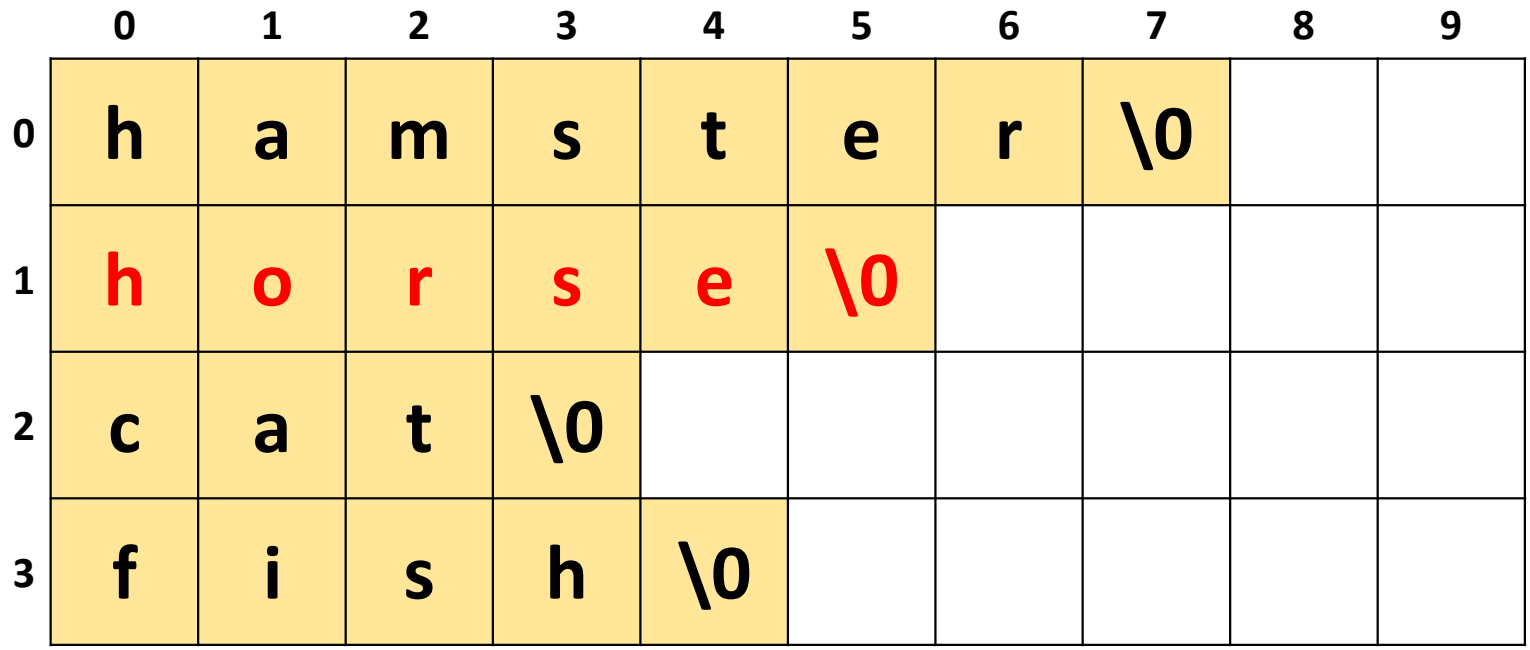
String Array Elements 8

What is the value of **names[1]**?

	0	1	2	3	4	5	6	7	8	9
0	h	a	m	s	t	e	r	\0		
1	h	o	r	s	e	\0				
2	c	a	t	\0						
3	f	i	s	h	\0					

String Array Elements 9

What is the value of **names[1]**? The string horse



CODE EXAMPLES

Fills list with user input directly via scanf

- listOfNamesUsingScanf.c

Fills list with user input via strcpy

- listOfNamesUsingstrcpy.c

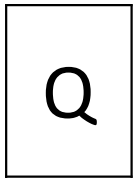
Displays contents of array via nested loops with single characters

- listOfNamesSingleChars.c

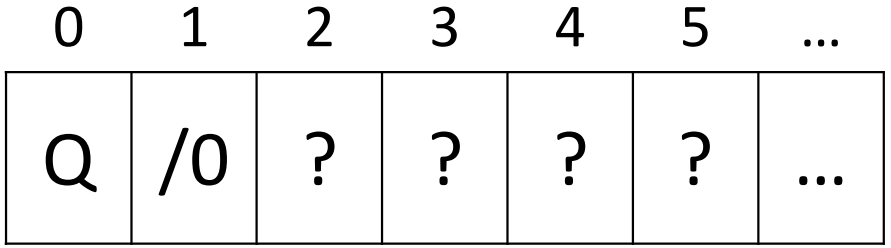
WORKING WITH CHARACTERS

CHARACTERS AND STRINGS

Not the same thing!



Character
with a value
'Q'



String
with a value
"Q"

CHARACTER OPERATIONS

We often must work with the **individual characters** that make up a string.

Character handling functions available in **ctype.h** library

CTYPE.H

Functions that perform useful tests and manipulations of character data

Characters are often manipulated as integers

- A character in C is a one-byte integer
- EOF normally has the value –1

CHARACTER INPUT WITH GETCHAR

getchar() returns a character (as its ASCII integer code) from stdin:

```
ch = getchar();
```

CHARACTER INPUT WITH SCANF

Can also use scanf:

scanf("%c", &ch)

But the values of these **expressions** are different…

HANDLING CHARACTER INPUT

Function	Example	Return Value of Expression
getchar()	ch = getchar();	**character** as its integer ASCII code
scanf	status = scanf("%c", &ch);	**count of items** read **side effect** is the character is stored in the variable ch

CHARACTER OUTPUT WITH PUTCHAR

putchar(*character*) outputs one character to stdout

putchar(*ASCII code*) outputs one character to stdout

Examples:

```
putchar('A');    //displays uppercase A

putchar(65);     //also displays uppercase A
```

CHARACTER ANALYSIS AND CONVERSION

Sometimes we need to know if a character is a **digit**, a **letter**, a **punctuation** mark, or part of some **other character set**.

Character classification functions are also in the ctype.h library

Also have functions that can convert the **case** of character, e.g. upper to lower

CHARACTER CLASSIFICATION FUNCTIONS 1

Function	Checks For	Example
isalpha	letter	if (isalpha (ch))
isdigit	digit	if (isdigit (ch))
isalnum	letter or digit	if (isalnum (ch))
isxdigit	hexadecimal digit	if (isxdigit (ch))
islower	lowercase letter	if (islower (ch))
isupper	uppercase letter	if (isupper (ch))

CHARACTER CLASSIFICATION FUNCTIONS 2

Function	Checks For	Example
ispunct	punctuation	if (ispunct (ch))
isspace	whitespace	if (isspace (ch))
iscntrl	tab, form feed, alert, backspace, carriage return or newline	if (iscntrl (ch))
isprint	printable character	if (isprint (ch))

CHARACTER CONVERSION FUNCTIONS

Function	Checks For	Example
tolower	lowercase letter	lower_c = tolower (upper_c);
toupper	uppercase letter	printf("Capital %c = %c", ch, toupper(ch);

EXAMPLES

Fig08_02.c

- Demonstrates functions **isdigit**, **isalpha**, **isalnum** and **isxdigit**
- Uses the **ternary conditional operator** (?:)

Fig08_03.c

- Demonstrates functions **islower**, **isupper**, **tolower** and **toupper**

Fig08_04.c

- Demonstrates functions **isspace**, **iscntrl**, **ispunct**, **isprint**, **isgraph**

String Summary

STRING SUMMARY 1

Sequence of characters in a single grouping

String variable declared by creating an **array of chars**

Use **double quotes** around **“**string constants**”**

Null terminator ends properly-formatted strings

Use **"%s"** as the placeholder for scanf and printf

scanf does not need **&** for string variables

Cannot assign to a string except at initialization

STRING SUMMARY 2

Need special **string library functions** in **<string.h>**

strcpy & strncpy are string assignment functions

Substring is a part of a larger string

Can **use strncpy to extract substrings** from larger strings

Concatenate means join

strcat & strncat are concatenation functions

Use **gets** function instead of scanf to read entire lines

strcmp & strncmp are string comparison functions