



C-10 Characters and Strings

Two vertical bars are located on the left side of the slide. The left bar is dark green and the right bar is yellow. Both bars are of equal height and width.

REMINDER:

Characters and Integers are closely related.

Declaring Character Variables

1. Single Characters
Stored in binary

Declared as type *char*

EX:

```
char name, a1 = 'a';  
int n1, n2;
```



Declaring Character Variables

2. Character Strings:

Several characters stored in an array of char

Stored in binary

By definition, ends with NULL

Examples on next slide...

Declaring Character Strings. Examples:

```
char my_name[12 ] = "Ruthann";
```

- C will recognize **my_name** as a string and add the NULLs at the end.
- Will consist of R,u,t,h,a,n,n,\0 ,\0 ,\0 ,\0 ,\0
 - **\0** is the same as **NULL**

```
char filename[ ] = "lab15.dat";
```

will default to a length of 9 + 1 for NULL = 10

```
char name[30];
```



Now to deal with getting
characters in and out....I/O.

Some choice.



Choice 1:

We can use **printf** and **scanf** for character I/O utilizing the **%c** specifier.

```
char letter;  
scanf ("%c", &letter);  
printf("The letter is: %c \n", letter);
```

Choice 2, more common:

We can use special character functions called **getchar** and **putchar**:

getchar – reads a character from the keyboard and returns an integer value

putchar – prints a character to the screen

The function prototypes for both of these functions
Included in *stdio.h*

Function prototypes:

```
int getchar(void);  
int putchar(int);
```

Example:

```
int c;  
c = getchar( );  
putchar(c);
```




End of File (EOF)

Defined in stdio.h:

```
#define EOF (-1)
```

```
/* This value is system dependent and will  
vary from system to system */
```

How to implement EOF at the keyboard:

In Unix/Linux environments, press:

control-d (^d)


It is configurable/changeable.

Two vertical bars are located on the left side of the slide. The left bar is dark green and the right bar is yellow. Both bars are of equal height and width, and are positioned side-by-side.

Now a program to

- read chars from keyboard
- echo back to screen.

It also counts number of chars entered



```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int c, count = 0;
    printf("Enter character: (^d to quit) \n");
    c = getchar( );
    count++;
    while (c != EOF)
    {
        putchar(c);
        c = getchar( );
        count++;
    }
    printf("%i characters printed. \n", count - 1);
    return EXIT_SUCCESS;
}
```

Run the program:

Enter character: (^D to quit)

abcd

abcd 5 chars including newline

z

z 2 chars including newline

1w2e3r \$

1w2e3r \$ 9 chars including newline and space

^d 1 char

16 characters printed.

The 16 does not include the EOF marker/character.



Character I/O from Keyboard (review)

1. To read characters from the keyboard and write to screen:

```
c = getchar( );  
putchar(c);
```



Character I/O from **Files**

2. To read characters from a file and write to a file:

```
FILE *infile, *outfile;
```

```
infile = fopen("in.dat", "r");  
outfile = fopen("out.dat", "w");
```

```
c = fgetc(infile);
```


```
fputc(outfile, c);
```



Character I/O from Files

3. To read chars from a file in a while loop:

```
while ((c = fgetc(infile)) != EOF)
{
    fputc(outfile, c);
}
```

```
/* This program counts words line by line */ Page 1 of 4
/* count_words.c */
```

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

```
#include <stdlib.h>
```

```
#define FILENAME "Text1.dat"
```

```
#define NEWLINE '\n'
```

```
int main(void)
```

```
{
```

```
    int line[100], k = 0, count = 0;
```

```
    FILE *text1;
```

```
    int word_cnt(int x[ ], int npts);
```

```
    text1 = fopen(FILENAME, "r");
```

```
        /* omit error check to save room on slide */
```

```
    ... /* more on next slide */
```

...

```
while ((line[k] = fgetc(text1)) != EOF)
{
    if (line[k] == NEWLINE)
    {
        if (k != 0)
        {
            count += word_cnt(line, k);
        }
        k = 0;
    }
    else
        k++;
}
... /* and more on next slide */
```

```
...  
    if (k != 0)  
    {  
        count += word_cnt(line, k);  
    }  
    printf("\n%i words read. \n\n", count);  
    return EXIT_SUCCESS;  
}
```

The function **word_cnt** is on next slide.

```
/* Function to count number of words in an integer array */
int word_cnt(int x[ ], int npts) {
    int count = 0, k = 0;
    char space = ' ';
    while (k < npts)
    {
        while ((k < npts) && (x[k] == space)) {
            k++;
        }
        if (k < npts) {
            count++;
        }
        while ((k < npts) && (x[k] != space)) {
            k++;
        }
    }
    return count;
}
```

Two vertical bars, one dark green and one yellow, are positioned on the left side of the slide.

Character Functions in ctype.h

To use these functions, add to your code:

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

Sample of one function

tolower(c) returns a lower case letter

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

```
.....
```

```
    int c;
```

```
    while ((c = getchar() ) != EOF)
```

```
        putchar(tolower(c) );
```

```
.....
```



Various character functions all found in **ctype.h**:

`tolower(c)` returns a lower case letter

`toupper(c)` returns an upper case letter

`isdigit(c)` if (digit) return nonzero
 else return zero

`islower(c)` if (lower case) return nonzero
 else return zero

`isupper(c)` if (upper case) return nonzero
 else return zero



Various character functions all found in **ctype.h**:

isalpha(c) if (alphabetic) return nonzero
 else return zero

isalnum(c) if (alphanumeric) return nonzero
 else return zero

iscntrl(c) if (control char) return nonzero
 else return zero

isgraph(c) if (printable) return nonzero
 else return zero

isprint(c) if (printable or space) return nonzero
 else return zero



Various character functions all found in **ctype.h**:

`ispunct(c)` if (printable, but not space, letter, or
 digit) return nonzero
 else return zero

`isspace(c)` if (white space: space, formfeed, nl,
 cr, horizontal or vertical tab)
 return nonzero
 else return zero

`isxdigit(c)` if (hexadecimal digit: 0 1 2 3 4 5 6 7
 8 9 A B C D E F a b c d e f)
 return nonzero
 else return zero

Two vertical bars, one dark green and one yellow, are positioned on the left side of the slide.

Character Arrays


Character Arrays

Character array – each letter stored as individual character element of the array

Character **string** – a character array where the last array element is NULL ('\0')

Character string constants are enclosed in **double quotes**. All three of these do the same thing:

```
char name[5] = "Jane";  
char name[ ] = "Jane";  
char name[ ] = {'J', 'a', 'n', 'e', '\0'};
```



We may get a **compilation error** if we try to put too many characters into an array of a defined length:

```
char name[6] = "Janice";
```

Some compilers give an error like this:

```
error C2117: 'Janice' : array bounds overflow
```

We need a space for the NULL, '\0', at the end.

Our compiler will NOT give an error, but the program may not run correctly.

```

#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    char a[6]="234";
    /* first see what is in the array
       after initialization */

    for (int j=0; j < 6; j++)
    {
        if (a[j] == NULL)
            printf("%i  Null\n", j);
        else
            printf("%i  \"%c\" \n", j, a[j]);
    }

    printf("\n\n");

    return EXIT_SUCCESS;
}

```

Output:

```

0  "2"
1  "3"
2  "4"
3  Null
4  Null
5  Null

```

```
/* Now change one value and look  
    again to see what is there */
```

```
a[4] = '9';  
for ( j=0; j < 6; j++)  
{  
    if (a[j] == NULL)  
        printf("%i  Null\n", j);  
    else  
        printf("%i  \"%c\" \n", j, a[j]);  
}  
return EXIT_SUCCESS;  
}  
/*-----*/
```

Output:

0	"2"
1	"3"
2	"4"
3	Null
4	"9"
5	Null



Output as it appeared on the screen:

/-----*/*

0 "2"

1 "3"

2 "4"

3 Null

4 Null

5 Null

0 "2"

1 "3"

2 "4"

3 Null

4 "9"

5 Null

Two vertical bars, one dark green and one yellow, are positioned on the left side of the slide.

C library string functions found in string.h:

To use these functions, add to your code:

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




```
size_t strlen(s);
```

```
// returns length of string s  
// that is filled with values.
```

```
/* ----- */
```

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

```
#include <stdlib.h>
```

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

```
int main(void)
```

```
{
```

```
    char a[6] = {"234"};
```

```
    int len;
```

```
    len = strlen(a);
```

```
    printf("\nLength = %i\n", len);
```

```
    return EXIT_SUCCESS;
```

```
}
```

```
/* ----- */
```

```
Length = 3
```

char *strcpy(s, t); copies string t onto and
over string s

```
/*-----*/  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
int main(void)  
{  
    char a[10] = {"234"};  
    char b[10] = {"ABC"};  
    strcpy(a, b);  
    printf("\nNew String A = %s\n", a);  
    return EXIT_SUCCESS;  
}  
/*-----*/
```

New String A = ABC

char *strncpy(s, t, n); copies n characters of string t to s

```
/*-----*/
```

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

```
#include <stdlib.h>
```

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

```
int main(void)
```

```
{
```

```
    char a[10] = {"234"};
```

```
    char b[10] = {"ABC"};
```

```
    strncpy(a, b, 2);
```

```
    printf("\nNew String A = %s\n", a);
```

```
    return EXIT_SUCCESS;
```

```
}
```

```
/*-----*/
```

New String A = AB4

char *strcat(s, t); concatenates string t to the end of s
(concatenates = to paste on end)

```
/*-----*/
```

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

```
#include <stdlib.h>
```

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

```
int main(void)
```

```
{
```

```
    char a[10] = {"234"};
```

```
    char b[10] = {"ABC"};
```

```
    strcat(a, b);
```

```
    printf("\nNew String A = %s\n", a);
```

```
    return EXIT_SUCCESS;
```

```
}
```

```
/*-----*/
```

New String A = 234ABC

char *strncat(s, t, n); concatenates n chars of t to end of s

```
/*-----*/  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
int main(void)  
{  
    char a[10] = {"234"};  
    char b[10] = {"ABC"};  
    strncat(a, b, 2);  
    printf("\nNew String A = %s\n", a);  
    return EXIT_SUCCESS;  
}  
/*-----*/
```

New String A = 234AB



int strcmp(s, t); compares strings s and t;

returns:

a negative value, if **s < t**,

zero, if **s == t**,

a positive value, if **s > t**

This function will be used in the mini-shell program.

```

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

int main(void) {
    int answer1, answer2;
    int answer3, answer4;

    char a[10] = {"234"};
    char b[10] = {"ABC"};
    char c[10] = {"ABC"};
    char d[10] = {"abc"};
    char e[10] = {"cba"};

    answer1 = strcmp(a, b);
    answer2 = strcmp(b, c);
    answer3 = strcmp(c, d);
    answer4 = strcmp(d, a);
    answer5 = strcmp(d, e);

```

```

printf("\n ab = %3i"
       "\n bc = %3i"
       "\n cd = %3i"
       "\n ad = %3i\n",
       "\n de = %3i\n",
       answer1, answer2,
       answer3, answer4,
       answer5);
return EXIT_SUCCESS;
}
/*-----*/

```

Output:

```

ab = -1
bc =  0
cd = -1
ad =  1
de = -1

```



int strncmp(s, t, n);

compares at most n chars of s to t;

returns:

a negative value, if **s < t**,

zero, if **s == t**,

a positive value, if **s > t**


```

/*-----*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
    int answer1, answer2;
    int answer3, answer4;

    char a[10] = {"234"};
    char b[10] = {"ABC"};
    char c[10] = {"ABC"};
    char d[10] = {"abc"};

    answer1 = strncmp(a, b, 1);
    answer2 = strncmp(b, c, 2);
    answer3 = strncmp(c, d, 2);
    answer4 = strncmp(d, a, 2);

```

```

printf("\n ab = %3i"
       "\n bc = %3i"
       "\n cd = %3i"

       "\n ad = %3i\n",
       answer1, answer2,
       answer3, answer4);
return EXIT_SUCCESS;
}
/*-----*/

```

Output:

```

ab = -1
bc =  0
cd = -1
da =  1

```

Two vertical bars, one dark green and one yellow, are positioned on the left side of the slide.

More String Functions

Using Pointer Notation



char *strchr(s, c);

returns a pointer to the first occurrence
of character c in the string s;

returns NULL if c does not occur in s

```
/*-----*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
    char *where_ptr;

    char a[10] = {"234"};
    where_ptr = strchr(a, '3');

    printf("\nDereference of where_ptr = %c\n",
           *where_ptr);
    return EXIT_SUCCESS;
}
/*-----*/

Dereference of where_ptr = 3
```



```
char * strrchr(s, c);
```

returns a pointer to the last occurrence
of character c in the string s;

returns NULL if c does not occur in s.

```

/*-----*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
    char *where_ptr;
    char a[10] = {"234234"};

    where_ptr = strchr(a, '3');
    printf("\nDereference of where_ptr = %c\n",
        *where_ptr);
    printf("Address of a[1]      = %u\n"
        "Address of a[4]      = %u\n"
        "Address in where_ptr = %u\n",
        &a[1], &a[4], where_ptr);
    return EXIT_SUCCESS;
}
/*-----*/

```

Output:

```

Dereference of where_ptr = 3
Address of a[1]      = 1245041
Address of a[4]      = 1245044
Address in where_ptr = 1245044

```



C library string functions found in **string.h**:

char *strstr(s, t);

returns a pointer to the start of the
string **t** within the string **s**;

returns NULL if **t** does not occur in **s**

```

/*-----*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
    char *where_ptr;
    char a[10] = {"234234"};
    where_ptr = strstr(a, "34");

    printf("\nDereference of where_ptr = %c\n",
           *where_ptr);
    printf("Address of a[1]      = %u\n"
           "Address of a[4]      = %u\n"
           "Address in where_ptr = %u\n",
           &a[1], &a[4], where_ptr);
    return EXIT_SUCCESS;
}
/*-----*/

```

Output:

```

Dereference of where_ptr = 3
Address of a[1]      = 1245041
Address of a[4]      = 1245044
Address in where_ptr = 1245041

```




The next two functions

- These next functions both search a null-terminated string **a** for occurrences of characters specified by whether they are include in a second null-terminated string **set**.



size_t strspn(s, set);

It searches the string **s** for the first occurrence of a character that is **not** included in the string **set**

The value returned is the length of the longest initial segment of **s** that consists of characters found in **set**.

If every character of **s** appears in **set**, then the total length of **s** (not counting the terminating null character) is returned.

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

int main(void)
{
    int set_len1, set_len2, set_len3;
    char a[10] = {"abcde"};
    char b[10] = {"bbcce"};
    char c[10] = {"cbbcca"};

    set_len1 = strstrn(a, "bc");
    set_len2 = strstrn(b, "bc");
    set_len3 = strstrn(c, "bc");

    printf("set_len1 = %i\n"
           "set_len2 = %i\n"
           "set_len3= %i\n",
           set_len1, set_len2, set_len3);
    return EXIT_SUCCESS;
}
```

Output:

```
set_len1 = 0
set_len2 = 4
set_len3 = 5
```



```
size_t strcspn(s, set);
```

Similar to **strspn** except that it searches **s** for the first occurrence of a character that **is** included in the string **set**, skipping over characters that are not in **set**.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(void)
{
    int set_len1, set_len2, set_len3;
    char a[10] = {"abcde"};
    char b[10] = {"bbcce"};
    char c[10] = {"asdcbbcca"};

    set_len1 = strcspn(a, "bc");
    set_len2 = strcspn(b, "bc");
    set_len3 = strcspn(c, "bc");

    printf("set_len1 = %i\n"
           "set_len2 = %i\n"
           "set_len3 = %i\n",
           set_len1, set_len2, set_len3);
    return EXIT_SUCCESS;
}
```

Output:

```
set_len1 = 1
set_len2 = 0
set_len3 = 3
```



char *strpbrk(s, t);

returns a pointer to the first occurrence in string s
of any character of string t;

returns NULL if none of the characters in t occur in s

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

int main(void)
{
    char *ptr;
    char a[10] = {"abcde"};

    ptr = strpbrk(a, "cd");

    printf("*pos1 = %c\n"
           "Address of a[2] = %u\n"
           "Address in ptr = %u\n",
           *ptr, &a[2], ptr);

    return EXIT_SUCCESS;
}
```

Output:

```
*pos1 = c
Address of a[2] = 1245042
Address in ptr  = 1245042
```



```
char *strtok(s, ct);
```

It extracts tokens from strings.

Searches string **s** for tokens delimited by characters from **ct**.

?returns NULL if none of the characters in **t** occur in **s**


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

int main(void)
{
    char *ptr;
    char s[15] = "abc,de,fghi";
    ptr = strtok(s, ",");
    while(ptr != NULL)
    {
        printf("Token = %s\n", ptr);
        ptr = strtok(NULL, ",");
    }
    return EXIT_SUCCESS;
}
```


In the loop, the call to strtok starts with a Null. This has strtok start from where it left off in the string, instead of repeatedly starting at the beginning.

Output:


Token = abc
Token = de
Token = fghi

ASCII Character Codes


<u>Dec</u>	<u>Oct</u>	<u>Hex</u>	<u>Chr</u>
000	000	00	NULL
001	001	01	SOH
002	002	02	STX, Start TX
003	003	03	ETX, End TX
004	004	04	EOT
005	005	05	ENQ, Inquire
006	006	06	ACK, Acknowledge
007	007	07	BEL, Bell
008	010	08	BS, Back Space
009	011	09	HT, Horizontal Tab
010	012	0A	LF, New Line(Line Feed)
011	013	0B	VT, Vertical Tab
012	014	0C	FF, Form Feed
013	015	0D	CR, Carriage Return




<u>Dec</u>	<u>Oct</u>	<u>Hex</u>	<u>Chr</u>
014	016	0E	SO, Stand Out
015	017	0F	SI, Stand In
016	020	10	DLE
017	021	11	DC1
018	022	12	DC2
019	023	13	DC3
020	024	14	DC4
021	025	15	NAK, Negative ACK
022	026	16	SYN
023	027	17	ETB
024	030	18	CAN
025	031	19	EM
026	032	1A	SUB
027	033	1B	ESC, Escape
028	034	1C	FS, Cursor Right
029	035	1D	GS, Cursor Left
030	036	1E	RS, Cursor Up
031	037	1F	US, Cursor Down
032	040	20	SP, Space




<u>Dec</u>	<u>Oct</u>	<u>Hex</u>	<u>Chr</u>
033	041	21	!
034	042	22	"
035	043	23	#
036	044	24	\$
037	045	25	%
038	046	26	&
039	047	27	'
040	050	28	(
041	051	29)
042	052	2A	*
043	053	2B	+
044	054	2C	,
045	055	2D	-
046	056	2E	,
047	057	2F	/




<u>Dec</u>	<u>Oct</u>	<u>Hex</u>	<u>Chr</u>
048	060	30	0
049	061	31	1
050	062	32	2
051	063	33	3
052	064	34	4
053	065	35	5
054	066	36	6
055	067	37	7
056	070	38	8
057	071	39	9
058	072	3A	:
059	073	3B	;
060	074	3C	<
061	156	66	n
062	076	3E	>
063	077	3F	?
064	100	40	@




<u>Dec</u>	<u>Oct</u>	<u>Hex</u>	<u>Chr</u>
065	101	41	A
066	102	42	B
067	103	43	C
068	104	44	D
069	105	45	E
070	106	46	F
071	107	47	G
072	110	48	H
073	111	49	I
074	112	4A	J
075	113	4B	K
076	114	4C	L
077	115	4D	M
078	116	4E	N
079	117	4F	O
080	120	50	P



<u>Dec</u>	<u>Oct</u>	<u>Hex</u>	<u>Chr</u>
081	121	51	Q
082	122	52	R
083	123	53	S
084	124	54	T
085	125	55	U
086	126	56	V
087	127	57	W
088	130	58	X
089	131	59	Y
090	132	5A	Z
091	133	5B	[
092	134	5C	\
093	135	5D]
094	136	5E	^
095	137	5F	_
096	140	60	`



<u>Dec</u>	<u>Oct</u>	<u>Hex</u>	<u>Chr</u>
097	141	61	a
098	142	62	b
099	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6A	j
107	153	6B	k
108	154	6C	l
109	155	6D	m
110	156	66	n
111	157	66	o
112	160	70	p
113	161	71	q



<u>Dec</u>	<u>Oct</u>	<u>Hex</u>	<u>Chr</u>
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7A	z
123	173	7B	{
124	174	7C	
125	175	7D	}
126	176	7E	~
127	177	7F	DEL, Delete



C-10 Characters and Strings

The End