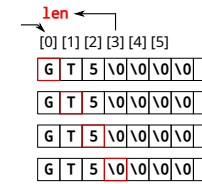
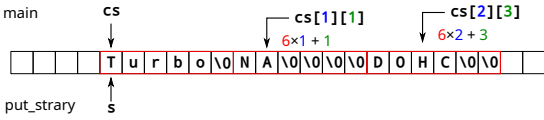



	もくじ	今回 (#06) の内容 : シラバスでの該当部分
<div> <div>#06 文字列の操作 2022 年度 / プログラミング及び実習 III</div> <div>角川裕次 龍谷大学 先端理工学部</div> </div> <div>1 / 47</div>	<div> <div>1 第 9-2 節 文字列の配列</div> <div>2 第 9-3 節 文字列の操作</div> <div>3 (独自) 文字に関する標準ライブラリ関数</div> <div>4 (独自) 文字列に関する標準ライブラリ関数</div> </div> <div>2 / 47</div>	<div>小テーマ: 文字列の操作 第 9 回 : 文字列のプログラミング</div> <div>3 / 47</div>
重要概念リスト	今回の実習・課題 (manaba へ提出)	
<div> <div>■ 文字列の長さ</div> <div>■ 文字列を表示する関数 : puts, fputs</div> <div>■ 書式付で表示する関数 : printf, fprintf</div> <div>■ 文字を表示する関数 : putc, fputc, putchar</div> <div>■ 大文字/小文字の変換 : toupper, tolower</div> </div> <div>4 / 47</div>	<div> <div>実習内容と課題内容は講義途中に提示します</div> <div>(作成したファイル類は manaba に提出)</div> </div> <div>5 / 47</div>	<div>第 9-2 節 文字列の配列</div> <div>6 / 47</div>

文字列の長さのプログラム例	文字列の長さ, プログラム (List 9-8) 実行例	関数 str_length() の観察
<p>List 9-8 : 文字列の長さを調べる</p> <pre>#include <stdio.h> int str_length(const char s[]) { int len = 0; while (s[len]) len++; return len; } int main(void) { char str[128]; printf("文字列を入力せよ:"); scanf("%s", str); printf("文字列\"%s\"の長さは%dです。\\n", str, str_length(str)); return 0; }</pre> <p>13 / 47</p>	<p>実行例 1</p> <pre>\$./list-9-8 文字列を入力せよ: ABC 文字列 "ABC" の長さは3です。</pre> <p>実行例 2</p> <pre>\$./list-9-8 文字列を入力せよ: JugemuJugemu 文字列 "JugemuJugemu" の長さは12です。</pre> <p>14 / 47</p>	<pre>int str_length(const char s[]) { int len = 0; while (s[len]) len++; return len; }</pre> <p>while (s[len]) は while (s[len] != '\0') と同じ</p>  <p>15 / 47</p>
<p>scanf の問題点がわかる実行例</p> <pre>\$./list-9-8 文字列を入力せよ: ABC ABC 文字列 "ABC" の長さは3です。</pre> <ul style="list-style-type: none">■ scanf : 空白を区切りとみなしている■ 空白を含む文字列を取り扱うには不向き■ これが scanf の使用をおすすめしない理由のひとつ <p>16 / 47</p>	<p>1 行読み込み (scanf 不使用版) の実装</p> <p>scanf の代わりに fgets (1 行読み込み) を使うのが良い</p> <ul style="list-style-type: none">■ EOF を読むと fgets() は NULL を返す■ fgets() は '\n' も含めて読み込む仕様: 除去が必要 <p>List 9-8 (改; 主要部) : 1 行の文字列を読み込む</p> <pre>char *read_string(int n, char *s) { char *p = fgets(s, n, stdin); if (p != NULL) { char *q = index(s, '\n'); if (q != NULL) { /* 文字列最後の改行 (Enterキー) は除去 */ *q = '\0'; } } return p; }</pre> <p>17 / 47</p>	<p>1 行読み込み (scanf 不使用版) の使用の main 関数</p> <pre>#include <stdio.h> #include <string.h> int main(void) { char str[8]; printf("文字列を入力せよ:"); if (read_string(sizeof(str), str) != NULL) { printf("文字列\"%s\"の長さは%dです。\\n", str, str_length(str)); } else { printf("END-OF-FILE\\n"); } return 0; }</pre> <p>注意 : 文字配列の要素数を 8 にしている</p> <ul style="list-style-type: none">■ わざと短くしている (問題発生を分かりやすくしている) <p>18 / 47</p>

<p>1 行読み込み (scanf 不使用版) : 実行例</p> <p>バッファオーバーフローを回避できている</p> <ul style="list-style-type: none"> ■ 要素数 8 の文字配列へ長い入力を与えても OK <pre>\$./list-9-8-fixed 文字列を入力せよ : Jugemu 文字列 "Jugemu" の長さは6です。</pre> <pre>\$./list-9-8-fixed 文字列を入力せよ : JugemuJugemu 文字列 "JugemuJ" の長さは7です。</pre> <p>文字列に空白が含まれていても期待通りに動作</p> <pre>\$./list-9-8-fixed 文字列を入力せよ : X Y Z 文字列 "X Y Z" の長さは5です。</pre> <p>19 / 47</p>	<p>文字列の表示 p.266</p> <p>表示のための関数 : putchar, fputc, putc, puts, fputs, printf, fprintf</p> <p>putchar 関数</p> <ul style="list-style-type: none"> ■ (標準出力に) 文字を出力 <p>fputc 関数, putc 関数</p> <ul style="list-style-type: none"> ■ (指定ストリームに) 文字を出力 <p>puts 関数</p> <ul style="list-style-type: none"> ■ (標準出力に) 文字列を出力 <p>fputs 関数</p> <ul style="list-style-type: none"> ■ (指定ストリームに) 文字列を出力 <p>printf 関数</p> <ul style="list-style-type: none"> ■ (標準出力に) 文字列, 文字, 整数, 実数等を書式に従って出力 <p>fprintf 関数</p> <ul style="list-style-type: none"> ■ (指定ストリームに) 文字列, 文字, 整数, 実数等を書式に従って出力 <p>20 / 47</p>	<p>putchar 関数を使って文字列を表示</p> <p>List 9-9 (部分) : putchar 関数を使って文字列を表示</p> <pre>void put_string(const char s[]) { int i = 0; while (s[i]) putchar(s[i++]); }</pre> <p>注 : while (s[i]) は while (s[i] != '\0') と同じ</p> <p>動作</p> <ul style="list-style-type: none"> ■ 文字列の先頭から 1 文字ずつ putchar 関数で表示してゆく ■ ただしナル文字に出会うと終了 ■ ナル文字は表示しない <p>21 / 47</p>
<p>数字文字の出現回数 p.267</p> <p>0 から 9 までの各数字が文字列中に何回出現するかを勘定</p> <p>実行例</p> <pre>文字列を入力せよ : pi=3.1415926535,e=2.71828 数字文字の出現回数 '0':0 '1':3 '2':3 '3':2 '4':1 '5':3 '6':1 '7':1 '8':2 '9':1</pre> <p>22 / 47</p>	<p>文字列中の各数字の出現回数のカウント方 : main 関数</p> <p>List 9-10 (部分 1/2) : 文字列の入力と結果表示</p> <pre>int main(void) { int dcnt[10] = {0}; /* 分布 */ char str[128]; /* 文字列 */ printf("文字列を入力せよ : "); scanf("%s", str); str_dcount(str, dcnt); puts("数字文字の出現回数"); for (int i = 0; i < 10; i++) printf("'d' : %d\n", i, dcnt[i]); return 0; }</pre> <p>23 / 47</p>	<p>数字文字の出現回数, 本体部分</p> <p>List 9-10 (部分 2/2) : 実際のカウンタ</p> <pre>void str_dcount(const char s[], int cnt[]) { int i = 0; while (s[i]) { if (s[i] >= '0' && s[i] <= '9') cnt[s[i] - '0']++; i++; } }</pre> <p>動作</p> <ul style="list-style-type: none"> ■ 文字列を先頭から 1 文字ずつ眺めてゆく ■ 数字 ('0', '1', ..., '9') なら 配列 cnt の対応する要素の値を 1 増加 <p>24 / 47</p>

<div>大文字・小文字の変換 p.268</div> <div>関数 toupper(int c) : 小文字を大文字に変換する<ul style="list-style-type: none">■ 小文字以外はそのまま■ 例 : a A■ 例 : A A■ 例 : 0 0■ ヘッダ ctype.h のインクルードが必要■ いわゆる半角文字だけが変換対象 (全角文字は対象外)List 9-11 (部分): 文字列中の小文字をすべて大文字に変換<pre>void str_toupper(char s[]) { int i = 0; while (s[i]) { s[i] = toupper(s[i]); i++; } }</pre></div> <div>25 / 47</div>	<div>大文字・小文字の変換 (つづき)</div> <div>関数 tolower(int c) : 大文字を小文字に変換する<ul style="list-style-type: none">■ 大文字以外はそのまま■ 例 : A a■ 例 : a a■ 例 : 0 0■ ヘッダ ctype.h のインクルードが必要■ いわゆる半角文字だけが変換対象 (全角文字は対象外)List 9-11 (部分): 文字列中の大文字をすべて小文字に変換<pre>void str_tolower(char s[]) { int i = 0; while (s[i]) { s[i] = tolower(s[i]); i++; } }</pre></div> <div>26 / 47</div>	<div>文字列の配列の受け渡し p.270</div> <div>List 9-12 : 文字列の配列の内容を表示する関数 (仮引数 s[][6] に注目)</div> <div><pre>#include <stdio.h> void put_strary(const char s[][6], int n) { for (int i = 0; i < n; i++) printf("s[%d] = \"%s\"\n", i, s[i]); } int main(void) { char cs[][6] = {"Turbo", "NA", "DOHC"}; put_strary(cs, 3); return 0; }</pre></div> <div>出力 (表示)</div> <div><pre>s[0] = "Turbo" s[1] = "NA" s[2] = "DOHC"</pre></div> <div>27 / 47</div>
<div>仮引数 char s[][6] の 6 は省略できない</div> <div><pre>void put_strary(const char s[][6], int n) { ... }</pre></div> <div>Q: 関数内で要素 s[i][j] の配置アドレスはどうやって求める?</div> <div>A: 6 という情報があれば要素の配置アドレスを (コンパイラは) 計算可能</div> <div>要素 s[i][j] の配置アドレス = s の先頭のアドレス + sizeof(char) × (6 × i + j)</div> <div></div> <div>6 という情報がないと要素の配置アドレスを計算する手立てがない (変数宣言を正しくすれば配置アドレス計算は気にする必要なし)</div> <div>28 / 47</div>	<div>Q. でも他で仮引数を char p[] とかやってましたよ?</div> <div>A. それができるのは 1 要素のサイズが char で分かるからです<ul style="list-style-type: none">■ なので p[10] の配置アドレスを計算できます</div> <div>Q. char s[][] でも 1 要素のサイズが分かりますよね?</div> <div>A. それだと char s[] のサイズが分かりません<ul style="list-style-type: none">■ s[2][3] の配置アドレスを計算できません</div> <div></div> <div>s[2][3] : 要素数 2 の配列<ul style="list-style-type: none">■ 各要素 : 要素数 3 の文字配列■ この 1 要素のサイズが分かる必要あり</div> <div>29 / 47</div>	<div>putchar 関数を使ってみる</div> <div>List 9-13 (部分) : 文字列の配列を表示</div> <div><pre>void put_strary2(const char s[][6], int n) { for (int i = 0; i < n; i++) { int j = 0; printf("s[%d] = \"", i); while (s[i][j]) putchar(s[i][j++]); puts("\"); } }</pre></div> <div>注 : while (s[i][j]) は while (s[i][j] != '\0') と同じ</div> <div>30 / 47</div>

	文字に関する標準ライブラリ関数：概要	文字種別の判定 (1)
<p>(独自) 文字に関する標準ライブラリ関数</p>	<p>インクルードすべきヘッダ</p> <pre>#include <ctype.h></pre> <p>わりとよく使う：</p> <p>isspace(c) — スペース文字か？</p> <p>isprint(c) — 表示可能文字 (スペース以外) か？</p> <p>isupper(c) — 大文字アルファベットか？</p> <p>islower(c) — 小文字アルファベットか？</p> <p>isalpha(c) — アルファベットか？</p> <p>isalnum(c) — アルファベットまたは数字 (10 進数) か？</p> <p>isdigit(c) — 数字 (10 進数) か？</p> <p>isxdigit(c) — 数字 (16 進数) か？</p>	<pre>int isspace(int c)</pre> <ul style="list-style-type: none"> c はスペースか否か 具体的にスペースとは：空白, '\f', '\n', '\r', '\t', '\v' <pre>! " # \$ % & ' () * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [\] ^ _ ` a b c d e f g h i j k l m n o p q r s t u v w x y z { } ~</pre> <pre>int isprint(int c)</pre> <ul style="list-style-type: none"> c は表示可能文字 (含むスペース) か否か <pre>! " # \$ % & ' () * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [\] ^ _ ` a b c d e f g h i j k l m n o p q r s t u v w x y z { } ~</pre>
31 / 47	32 / 47	33 / 47
文字種別の判定 (2)	文字種別の判定 (3)	文字種別の判定 (4)
<pre>int isupper(int c)</pre> <ul style="list-style-type: none"> c は大文字アルファベットか否か <pre>! " # \$ % & ' () * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [\] ^ _ ` a b c d e f g h i j k l m n o p q r s t u v w x y z { } ~</pre> <pre>int islower(int c)</pre> <ul style="list-style-type: none"> c は小文字アルファベットか否か <pre>! " # \$ % & ' () * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [\] ^ _ ` a b c d e f g h i j k l m n o p q r s t u v w x y z { } ~</pre>	<pre>int isalpha(int c)</pre> <ul style="list-style-type: none"> c はアルファベットか否か (isupper(c) islower(c)) と等価 <pre>! " # \$ % & ' () * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [\] ^ _ ` a b c d e f g h i j k l m n o p q r s t u v w x y z { } ~</pre> <pre>int isalnum(int c)</pre> <ul style="list-style-type: none"> c は英数字 (アルファベット又は数字) か否か (isalpha(c) isdigit(c)) と等価 <pre>! " # \$ % & ' () * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [\] ^ _ ` a b c d e f g h i j k l m n o p q r s t u v w x y z { } ~</pre>	<pre>int isdigit(int c)</pre> <ul style="list-style-type: none"> c は数字か否か <pre>! " # \$ % & ' () * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [\] ^ _ ` a b c d e f g h i j k l m n o p q r s t u v w x y z { } ~</pre> <pre>int isxdigit(int c)</pre> <ul style="list-style-type: none"> c は 16 進数表示用の文字か否か <pre>! " # \$ % & ' () * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [\] ^ _ ` a b c d e f g h i j k l m n o p q r s t u v w x y z { } ~</pre>
34 / 47	35 / 47	36 / 47

<div>文字種別の判定 (5)</div> <div><pre>int ispunct(int c) ■ c はスペース・英数文字以外か否か (記号か否か) !"#\$%&'()*+,-./ 0123456789:;<=>? @ABCDEFGHIJKLMNO PQRSTUVWXYZ[\]^_ `abcdefghijklmnopqrstuvwxyz { } ~</pre><pre>int isgraph(int c) ■ c は (スペース以外の) 表示可能な文字か否か !"#\$%&'()*+,-./ 0123456789:;<=>? @ABCDEFGHIJKLMNO PQRSTUVWXYZ[\]^_ `abcdefghijklmnopqrstuvwxyz { } ~</pre></div> <div>37 / 47</div>	<div>文字種別の判定 (そのほか)</div> <div><pre>int isascii(int c) ■ c は符号無し 7 ビットで表現可能な文字か否か int isblank(int c) ■ c は空白文字・タブ文字か否か int iscntrl(int c) ■ c は制御文字か否か</pre></div> <div>38 / 47</div>	<div>大文字/小文字変換</div> <div><pre>int toupper(int c) ■ 小文字は大文字に変換 (それ以外の文字はそのまま) toupper('a') 'A' toupper('X') 'X' int tolower(int c) ■ 大文字は小文字に変換 (それ以外の文字はそのまま) tolower('A') 'a' tolower('y') 'y'</pre></div> <div>39 / 47</div>
	<div>文字列に関する標準ライブラリ関数：概要</div> <div><div>インクルードすべきヘッダ</div><div>#include <string.h></div><div>strlen(s) — 長さ strcmp(s1,s2) — 大小比較 strncmp(s1,s2,n) — 大小比較 (文字数限定) strchr(s,c) — 文字 c の検索 (先頭から) strrchr(s,c) — 文字 c の検索 (末尾から) strstr(s,t) — 文字中の文字列の検索 strncat(s,t,n) — s の後に t を連結 strncpy(s,t,n) — s に t を上書きコピー strdup(s) — s のコピーを生成</div></div> <div>40 / 47</div>	<div>文字列の長さ</div> <div><pre>size_t strlen(const char *s) ■ 文字列 s の長さを返す</pre></div> <div>41 / 47</div>
<div>(独自) 文字列に関する標準ライブラリ関数</div> <div>42 / 47</div>		

文字列の比較	文字の検索	文字列の連結・コピー
<pre>int strcmp(const char *s1, const char *s2)</pre> <ul style="list-style-type: none"> ■ 文字列 <code>s1</code> と <code>s2</code> を比較 ■ 返り値: 0 (<code>s1 == s2</code> の場合) ■ 返り値: < 0 (<code>s1 < s2</code> の場合) ■ 返り値: > 0 (<code>s1 > s2</code> の場合) <pre>int strncmp(const char *s1, const char *s2, size_t n)</pre> <ul style="list-style-type: none"> ■ 文字列 <code>s1</code> と <code>s2</code> を比較 (先頭から <code>n</code> 文字まで) ■ 返り値: 0 (<code>s1 == s2</code> の場合) ■ 返り値: < 0 (<code>s1 < s2</code> の場合) ■ 返り値: > 0 (<code>s1 > s2</code> の場合) <div>43 / 47</div>	<pre>char *strchr(const char *s, int c)</pre> <ul style="list-style-type: none"> ■ 文字列 <code>s</code> の中の文字 <code>c</code> を検索 ■ 検索は文字列の先頭から ■ 見つければその文字へのポインタを返す ■ 見つからなければ NULL ポインタを返す <pre>char *strrchr(const char *s, int c)</pre> <ul style="list-style-type: none"> ■ 文字列 <code>s</code> の中の文字 <code>c</code> を検索 ■ 検索は文字列の末尾から ■ 見つければその文字へのポインタを返す ■ 見つからなければ NULL ポインタを返す <div>44 / 47</div>	<pre>char *strncat(char *dest, const char *src, size_t n)</pre> <ul style="list-style-type: none"> ■ 文字列 <code>dest</code> の末尾に文字列 <code>src</code> の内容を連結 ■ <code>dest</code> の長さの制限は <code>n</code> <pre>char *strncpy(char *dest, const char *src, size_t n)</pre> <ul style="list-style-type: none"> ■ 文字列 <code>dest</code> に文字列 <code>src</code> をコピー ■ コピーするのは <code>src</code> の <code>n</code> 文字まで ■ 要注意: <code>n</code> 文字ちょうどのコピーの場合 <code>dest</code> は \0 で終端されない <pre>char *strdup(const char *s)</pre> <ul style="list-style-type: none"> ■ 新たな文字配列 (メモリ) を割り当て文字列 <code>s</code> をコピー ■ 新たな文字配列へのポインタを返す ■ (その文字列が不要になれば <code>free()</code> でメモリ解放が必要) <div>45 / 47</div>
<div>おわり</div> <div>46 / 47</div>	<div>番外編の課題 1</div> <p>任意に与えられた文字列に対し文字列中の 1 単語を 1 行ごとに表示する関数の作成</p> <ul style="list-style-type: none"> ■ <code>void pstrword(const char *s);</code> ■ ただし単語を構成する文字はアルファベットと数字に限定 <p>例: <code>pstrword("He was born in 2001.");</code> の出力</p> <pre>He was born in 2001</pre> <p>例: <code>pstrword(" Hello, world. You have mail. ");</code> の出力</p> <pre>Hello world You have mail</pre> <div>47 / 47</div>	