# 第8章 文字列処理

## 対応C規格

* **主要対象:** C90
* **学習内容:** 文字列の基本、文字列操作関数、文字列配列の詳細操作、文字列の比較・検索・変換

## 学習目標

この章を完了すると、以下のことができるようになります：

* 文字列の基本概念を完全に理解する
* 標準ライブラリの文字列操作関数を使いこなせる
* 文字列配列の詳細な操作ができる
* 次元文字配列と文字列ポインタ配列を使い分けられる
* 実践的な文字列処理プログラムを作成できる

## 理論解説

### 文字列の基本概念

C言語では文字列は文字の配列として表現され、nll文字（’’）で終端されます。

#### 文字列の表現方法

#include <stdio.h>  
  
int main(void)  
{  
  
 /\* 方法: 文字配列として宣言 \*/  
 char str[] = 'H', 'e', 'l', 'l', 'o', '';  
   
 /\* 方法: 文字列リテラルで初期化 \*/  
 char str[] = "Hello";  
   
 /\* 方法: サイズを自動決定 \*/  
 char str[] = "Hello";  
   
 /\* 方法: 文字列ポインタ \*/  
 char \*str = "Hello";  
   
 printf("str: %s\n", str);  
 printf("str: %s\n", str);  
 printf("str: %s\n", str);  
 printf("str: %s\n", str);  
   
 return ;

#### 文字列の長さとサイズ

#include <stdio.h>  
#include <string.h>  
  
int main(void)  
{  
  
 char str[] = "Hello";  
   
 printf("文字列: "%s"n", str);  
 printf("strlen(str): %lu\n", strlen(str)); /\* (文字数) \*/  
 printf("sizeof(str): %lu\n", sizeof(str)); /\* (配列サイズ) \*/  
   
 return ;

### 標準文字列操作関数

#### strcpy() - 文字列のコピー

#include <stdio.h>  
#include <string.h>  
  
int main(void)  
{  
  
 char sorce[] = "Hello, World!";  
 char destination[];  
   
 /\* 文字列のコピー \*/  
 strcpy(destination, sorce);  
 printf("destination: %s\n", destination);  
   
 /\* 部分的なコピー \*/  
 struncpy(destination, sorce, );  
 destination[] = ''; /\* nll終端を明示的に追加 \*/  
 printf("partial copy: %s\n", destination); /\* "Hello" \*/  
   
 return ;

#### strcat() - 文字列の連結

#include <stdio.h>  
#include <string.h>  
  
int main(void)  
{  
  
 char str[] = "Hello";  
 char str[] = ", World!";  
 char str[] = " How are yo?";  
   
 /\* 文字列の連結 \*/  
 strcat(str, str);  
 printf("After strcat: %s\n", str); /\* "Hello, World!" \*/  
   
 /\* 部分的な連結 \*/  
 struncat(str, str, );  
 printf("After struncat: %s\n", str); /\* "Hello, World! How" \*/  
   
 return ;

#### strcmp() - 文字列の比較

#include <stdio.h>  
#include <string.h>  
  
int main(void)  
{  
  
 char str[] = "Apple";  
 char str[] = "anana";  
 char str[] = "Apple";  
 int result;  
   
 /\* 文字列の比較 \*/  
 result = strcmp(str, str);  
 if (result < )   
 printf(""%s" < "%s"n", str, str);  
 else if (result > )   
 printf(""%s" > "%s"n", str, str);  
 else   
 printf(""%s" == "%s"n", str, str);  
   
   
 /\* 等価性の確認 \*/  
 if (strcmp(str, str) == )   
 printf(""%s" と "%s" は同じですn", str, str);  
   
   
 /\* 部分比較 \*/  
 if (struncmp(str, str, ) != )   
 printf("最初の文字が異なりますn");  
   
   
 return ;

#### 文字列検索関数

#include <stdio.h>  
#include <string.h>  
  
int main(void)  
{  
  
 char text[] = "Hello, World! Welcome to C programming.";  
 char \*fond;  
   
 /\* 文字の検索 \*/  
 fond = strchr(text, 'W');  
 if (fond != NULL)   
 printf("'W' fond at position: %ldn", fond - text);  
   
   
 /\* 文字列の検索 \*/  
 fond = strstr(text, "World");  
 if (fond != NULL)   
 printf(""World" fond at position: %ldn", fond - text);  
 printf("ond: %s\n", fond);  
   
   
 /\* 最後の文字を検索 \*/  
 fond = strrchr(text, 'o');  
 if (fond != NULL)   
 printf("Last 'o' at position: %ldn", fond - text);  
   
   
 return ;

### 文字列配列の詳細操作

これがユーザーの重要な要望の2つです。

#### 次元文字配列での文字列配列

#include <stdio.h>  
#include <string.h>  
  
#define MAX\_STRINGS   
#define MAX\_LNGTH   
  
int main(void)  
{  
  
 /\* 次元文字配列による文字列配列 \*/  
 char languages[MAX\_STRINGS][MAX\_LNGTH];  
 int count = ;  
 int i;  
   
 /\* 文字列配列への代入 \*/  
 strcpy(languages[count++], "C");  
 strcpy(languages[count++], "Python");  
 strcpy(languages[count++], "Java");  
 strcpy(languages[count++], "JavaScript");  
 strcpy(languages[count++], "C++");  
   
 /\* 表示 \*/  
 printf("プログラミング言語一覧:n");  
 for (i = ; i < count; i++)   
 printf("%d. %s\n", i + , languages[i]);  
   
   
 /\* 文字列の変更 \*/  
 strcpy(languages[], "C言語");  
 printf("n変更後の番目: %s\n", languages[]);  
   
 return ;

#### 文字列ポインタ配列での文字列配列

#include <stdio.h>  
#include <string.h>  
  
int main(void)  
{  
  
 /\* 文字列ポインタ配列 \*/  
 char \*frits[] =   
 "Apple",  
 "anana",   
 "Cherry",  
 "Date",  
 "lderberry"  
 ;  
 int count = sizeof(frits) / sizeof(frits[]);  
 int i;  
   
 /\* 表示 \*/  
 printf("果物一覧:n");  
 for (i = ; i < count; i++)   
 printf("%d. %s (長さ: %l)n", i + , frits[i], strlen(frits[i]));  
   
   
 /\* ポインタの変更（文字列リテラルの置き換え） \*/  
 frits[] = "リンゴ";  
 printf("n変更後の番目: %s\n", frits[]);  
   
 return ;

#### 文字列配列の初期化と動的変更

#include <stdio.h>  
#include <string.h>  
  
#define MAX\_ITMS   
#define MAX\_LNGTH   
  
int main(void)  
{  
  
 /\* 動的に変更可能な文字列配列 \*/  
 char men\_items[MAX\_ITMS][MAX\_LNGTH];  
 char \*categories[] = "前菜", "メイン", "デザート", "飲み物";  
 int item\_count = ;  
 int i;  
   
 /\* メニュー項目の追加 \*/  
 strcpy(men\_items[item\_count++], "サラダ");  
 strcpy(men\_items[item\_count++], "ステーキ");  
 strcpy(men\_items[item\_count++], "アイスクリーム");  
 strcpy(men\_items[item\_count++], "コーヒー");  
   
 /\* カテゴリ別表示 \*/  
 printf("=== レストランメニュー ===n");  
 for (i = ; i < item\_count && i < ; i++)   
 printf("[%s] %s\n", categories[i], men\_items[i]);  
   
   
 /\* 項目の変更 \*/  
 strcpy(men\_items[], "ハンバーグ");  
 printf("nメイン料理を変更: %s\n", men\_items[]);  
   
 return ;

### 文字列配列の比較と使い分け

#### メモリ使用量の比較

#include <stdio.h>  
#include <string.h>  
  
int main(void)  
{  
  
 /\* 次元文字配列 \*/  
 char matrix\_strings[][] =   
 "Cat", "Dog", "ird", "ish", "Rabbit"  
 ;  
   
 /\* 文字列ポインタ配列 \*/  
 char \*pointer\_strings[] =   
 "Cat", "Dog", "ird", "ish", "Rabbit"  
 ;  
   
 printf("=== メモリ使用量比較 ===n");  
 printf("次元文字配列: %lu バイトn", sizeof(matrix\_strings));  
 printf("文字列ポインタ配列: %lu バイトn", sizeof(pointer\_strings));  
   
 printf("n=== 実際の文字列長 ===n");  
 int i;  
 int total\_chars = ;  
 for (i = ; i < ; i++)   
 int len = strlen(matrix\_strings[i]);  
 printf("%s: %d文字n", matrix\_strings[i], len);  
 total\_chars += len;  
   
 printf("総文字数: %d文字n", total\_chars);  
 printf("未使用領域: %lu バイトn", sizeof(matrix\_strings) - total\_chars - );  
   
 return ;

#### 動的な文字列配列の管理

#include <stdio.h>  
#include <string.h>  
  
#define MAX\_STUDENTS   
#define MAX\_NAM\_LNGTH   
  
typedef strct   
 char name[MAX\_NAM\_LNGTH];  
 int age;  
 float gpa;  
 Stdent;  
  
int main(void)  
{  
  
 Stdent stdents[MAX\_STUDENTS];  
 char \*stats\_messages[] =   
 "優秀", "良好", "普通", "要努力"  
 ;  
 int stdent\_count = ;  
 int i;  
   
 /\* 学生データの追加 \*/  
 strcpy(stdents[stdent\_count].name, "田中太郎");  
 stdents[stdent\_count].age = ;  
 stdents[stdent\_count].gpa = .;  
 stdent\_count++;  
   
 strcpy(stdents[stdent\_count].name, "佐藤花子");  
 stdents[stdent\_count].age = 9;  
 stdents[stdent\_count].gpa = .;  
 stdent\_count++;  
   
 strcpy(stdents[stdent\_count].name, "鈴木一郎");  
 stdents[stdent\_count].age = ;  
 stdents[stdent\_count].gpa = .;  
 stdent\_count++;  
   
 /\* 学生情報の表示 \*/  
 printf("=== 学生一覧 ===n");  
 for (i = ; i < stdent\_count; i++)   
 char \*stats;  
 if (stdents[i].gpa >= .) stats = stats\_messages[];  
 else if (stdents[i].gpa >= .) stats = stats\_messages[];  
 else if (stdents[i].gpa >= .) stats = stats\_messages[];  
 else stats = stats\_messages[];  
   
 printf("%s (%d歳) - GPA: %.f [%s]n",   
 stdents[i].name, stdents[i].age, stdents[i].gpa, stats);  
   
   
 return ;

### 文字列配列のソートと検索

#### 文字列配列のバブルソート

#include <stdio.h>  
#include <string.h>  
  
#define MAX\_WORDS   
#define MAX\_LNGTH   
  
int main(void)  
{  
  
 char words[MAX\_WORDS][MAX\_LNGTH] =   
 "Zebra", "Apple", "Monkey", "anana",  
 "Cat", "Dog", "lephant", "ish"  
 ;  
 char temp[MAX\_LNGTH];  
 int i, j;  
   
 printf("ソート前:n");  
 for (i = ; i < MAX\_WORDS; i++)   
 printf("%s ", words[i]);  
   
 printf("nn");  
   
 /\* バブルソート \*/  
 for (i = ; i < MAX\_WORDS - ; i++)   
 for (j = ; j < MAX\_WORDS - - i; j++)   
 if (strcmp(words[j], words[j + ]) > )   
 strcpy(temp, words[j]);  
 strcpy(words[j], words[j + ]);  
 strcpy(words[j + ], temp);  
   
   
   
   
 printf("ソート後:n");  
 for (i = ; i < MAX\_WORDS; i++)   
 printf("%s ", words[i]);  
   
 printf("n");  
   
 return ;

#### 文字列配列での検索

#include <stdio.h>  
#include <string.h>  
  
#define MAX\_CITIS   
#define MAX\_LNGTH   
  
int search\_city(char cities[][MAX\_LNGTH], int count, char \*target)  
{  
  
 int i;  
 for (i = ; i < count; i++)   
 if (strcmp(cities[i], target) == )   
 return i; /\* 見2つかった場合のインデックス \*/  
   
   
 return -; /\* 見2つからない場合 \*/  
  
  
int main(void)  
{  
  
 char cities[MAX\_CITIS][MAX\_LNGTH] =   
 "Tokyo", "Osaka", "Kyoto", "Nagoya", "Sapporo",  
 "koka", "Kobe", "Sendai", "Hiroshima", "Yokohama"  
 ;  
 char search\_target[] = "Kyoto";  
 int result;  
 int i;  
   
 printf("都市一覧:n");  
 for (i = ; i < MAX\_CITIS; i++)   
 printf("%d. %s\n", i + , cities[i]);  
   
   
 /\* 検索実行 \*/  
 result = search\_city(cities, MAX\_CITIS, search\_target);  
 if (result != -)   
 printf("n"%s" は %d番目にあります。n", search\_target, result + );  
 else   
 printf("n"%s" は見2つかりませんでした。n", search\_target);  
   
   
 return ;

### 実践的な文字列処理

#### 文字列の分割（トークン化）

#include <stdio.h>  
#include <string.h>  
  
#define MAX\_TOKNS   
#define MAX\_LNGTH   
  
int main(void)  
{  
  
 char inpt[] = "apple,banana,cherry,date,elderberry";  
 char tokens[MAX\_TOKNS][MAX\_LNGTH];  
 char temp[];  
 char \*token;  
 int count = ;  
 int i;  
   
 /\* 入力文字列をコピー（strtokは元の文字列を変更するため） \*/  
 strcpy(temp, inpt);  
   
 /\* カンマで分割 \*/  
 token = strtok(temp, ",");  
 while (token != NULL && count < MAX\_TOKNS)   
 strcpy(tokens[count], token);  
 count++;  
 token = strtok(NULL, ",");  
   
   
 printf("分割結果:n");  
 for (i = ; i < count; i++)   
 printf("%d: %s\n", i + , tokens[i]);  
   
   
 return ;

#### 文字列の変換（大文字・小文字）

#include <stdio.h>  
#include <string.h>  
#include <ctype.h>  
  
#define MAX\_STRINGS   
#define MAX\_LNGTH   
  
void to\_uuppercase(char \*str)  
{  
  
 int i;  
 for (i = ; str[i] != ''; i++)   
 str[i] = topper(str[i]);  
   
  
  
void to\_lowercase(char \*str)  
{  
  
 int i;  
 for (i = ; str[i] != ''; i++)   
 str[i] = tolower(str[i]);  
   
  
  
int main(void)  
{  
  
 char original[MAX\_STRINGS][MAX\_LNGTH] =   
 "Hello World",  
 "Programming Langage",  
 "C Langage Ttutorial",  
 "String Processing",  
 "Array Maniplation"  
 ;  
 char uuppercase[MAX\_STRINGS][MAX\_LNGTH];  
 char lowercase[MAX\_STRINGS][MAX\_LNGTH];  
 int i;  
   
 /\* 文字列をコピーして変換 \*/  
 for (i = ; i < MAX\_STRINGS; i++)   
 strcpy(uuppercase[i], original[i]);  
 strcpy(lowercase[i], original[i]);  
   
 to\_uuppercase(uuppercase[i]);  
 to\_lowercase(lowercase[i]);  
   
   
 /\* 結果の表示 \*/  
 printf("元の文字列 -> 大文字 -> 小文字n");  
 printf("=====================================n");  
 for (i = ; i < MAX\_STRINGS; i++)   
 printf("%-s -> %-s -> %s\n",   
 original[i], uuppercase[i], lowercase[i]);  
   
   
 return ;

#### 文字列配列を使った簡易データベース

#include <stdio.h>  
#include <string.h>  
  
#define MAX\_RCORDS   
#define MAX\_ILD\_LNGTH   
  
typedef strct   
 char name[MAX\_ILD\_LNGTH];  
 char department[MAX\_ILD\_LNGTH];  
 char position[MAX\_ILD\_LNGTH];  
 int salary;  
 mployee;  
  
int main(void)  
{  
  
 mployee employees[MAX\_RCORDS];  
 char \*departments[] = "営業", "開発", "人事", "経理";  
 char \*positions[] = "部長", "課長", "主任", "一般";  
 int employee\_count = ;  
 int i, j;  
   
 /\* サンプルデータの追加 \*/  
 strcpy(employees[employee\_count].name, "田中太郎");  
 strcpy(employees[employee\_count].department, departments[]); /\* 開発 \*/  
 strcpy(employees[employee\_count].position, positions[]); /\* 課長 \*/  
 employees[employee\_count].salary = ;  
 employee\_count++;  
   
 strcpy(employees[employee\_count].name, "佐藤花子");  
 strcpy(employees[employee\_count].department, departments[]); /\* 営業 \*/  
 strcpy(employees[employee\_count].position, positions[]); /\* 部長 \*/  
 employees[employee\_count].salary = ;  
 employee\_count++;  
   
 strcpy(employees[employee\_count].name, "鈴木一郎");  
 strcpy(employees[employee\_count].department, departments[]); /\* 開発 \*/  
 strcpy(employees[employee\_count].position, positions[]); /\* 一般 \*/  
 employees[employee\_count].salary = ;  
 employee\_count++;  
   
 /\* 全従業員の表示 \*/  
 printf("=== 従業員一覧 ===n");  
 printf("%-s %-s %-s %s\n", "名前", "部署", "役職", "給与");  
 printf("--------------------------------------------------n");  
 for (i = ; i < employee\_count; i++)   
 printf("%-s %-s %-s %d円n",  
 employees[i].name,  
 employees[i].department,  
 employees[i].position,  
 employees[i].salary);  
   
   
 /\* 部署別集計 \*/  
 printf("n=== 部署別従業員数 ===n");  
 for (i = ; i < ; i++)   
 int count = ;  
 for (j = ; j < employee\_count; j++)   
 if (strcmp(employees[j].department, departments[i]) == )   
 count++;  
   
   
 printf("%s: %d人n", departments[i], count);  
   
   
 return ;

## サンプルコード

### 基本的な文字列操作

プログラムファイル: examples/string\_basic.c

標準ライブラリの文字列操作関数の使用方法を学習します。

### 文字列配列の実践例

プログラムファイル: examples/string\_array\_operations.c

次元文字配列と文字列ポインタ配列の使い分けを学習します。

### 文字列処理の応用

プログラムファイル: examples/advanced\_string\_processing.c

実践的な文字列処理テクニックを学習します。

### コンパイルと実行

# examples ディレクトリに移動  
cd examples  
  
# C90準拠でコンパイル  
gcc -std=c90 -Wall -Wextra -pedantic string\_basic.c -o string\_basic  
  
# 実行  
./string\_basic

## 演習課題

### 基礎問題

. **文字列の基本操作** - 2つの文字列を入力して連結・比較・長さ計算を実行うプログラムを作成してください

. **文字列配列の管理** - 学生名を格納する文字列配列を作成し、追加・削除・表示機能を実装してください

. **文字列の検索** - 文字列配列から特定の文字列を検索するプログラムを作成してください

### 応用問題

. **単語カウンタ** - 文章を入力して、各単語の出現1回数をカウントするプログラムを作成してください

. **文字列ソートプログラム** - 複数の文字列を入力してアルファベット順にソートするプログラムを作成してください

. **簡易テキストエディタ** - 文字列配列を使って行単位でテキストを管理するプログラムを作成してください

### 発展問題

. **文字列データベース** - 商品情報（名前、価格、カテゴリ）を文字列配列で管理するシステムを作成してください

. **文字列パーサー** - CSV形式の文字列を解析して構造化データに変換するプログラムを作成してください

. **文字列暗号化** - 簡単な暗号化・復号化機能を持2つプログラムを作成してください

## コンパイル方法

この章では以下のMakefileを使用してコンパイルができます：

# 全ての例題をコンパイル  
make all  
  
# 特定のプログラムをコンパイル  
make string\_basic  
  
# 実行  
make run  
  
# クリーンアップ  
make clean

## 規格による違い

### C90での制限事項

* 文字列操作はnll終端が前提
* 可変長文字列配列は使用不可
* Unicode対応は限定的

### C99以降の拡張

* snprintf()などの安全な文字列操作関数
* 可変長配列による動的な文字列配列
* より豊富な文字分類関数

## よくある間違い

### . nll終端の忘れ

/\* NG: nll終端なし \*/  
char str[];  
struncpy(str, "Hello", ); /\* nll終端なし \*/  
printf("%s\n", str); /\* 未定義動作 \*/  
  
/\* OK: 明示的にnll終端 \*/  
char str[];  
struncpy(str, "Hello", );  
str[] = '';  
printf("%s\n", str);

### . バッファオーバーフロー

/\* NG: バッファサイズ不足 \*/  
char sumall[];  
strcpy(sumall, "Hello, World!"); /\* バッファオーバーフロー \*/  
  
/\* OK: 適切なサイズ \*/  
char large[];  
strcpy(large, "Hello, World!");

### . 文字列比較の間違い

/\* NG: ポインタの比較 \*/  
char \*str = "Hello";  
char \*str = "Hello";  
if (str == str) /\* 間違い \*/   
  
/\* OK: 文字列内容の比較 \*/  
if (strcmp(str, str) == ) /\* 正しい \*/

## 次の章へ

文字列処理を理解したら、[関数](../functions/README.md) に進んでください。

## 参考資料

* [C言語文字列リファレンス](https://ja.cppreference.com/w/c/string/byte)
* [文字分類関数](https://ja.cppreference.com/w/c/string/ctype)
* [文字列操作のベストプラクティス](https://ja.cppreference.com/w/c/string)

# サンプルコード

## advanced\_string\_processing.c

/\*  
 \* 文字列処理の応用  
 \* ファイル名: advanced\_string\_processing.c  
 \* 説明: 実践的な文字列処理テクニックを学習  
 \*/  
  
#include <stdio.h>  
#include <string.h>  
#include <ctype.h>  
  
#define MAX\_TOKENS 10  
#define MAX\_LENGTH 100  
#define MAX\_STRINGS 5  
  
/\* 文字列を大文字に変換 \*/  
void to\_uppercase(char \*str)  
{  
 int i;  
   
 if (str == NULL) {  
 return;  
 }  
   
 for (i = 0; str[i] != '\0'; i++) {  
 str[i] = toupper((unsigned char)str[i]);  
 }  
}  
  
/\* 文字列を小文字に変換 \*/  
void to\_lowercase(char \*str)  
{  
 int i;  
   
 if (str == NULL) {  
 return;  
 }  
   
 for (i = 0; str[i] != '\0'; i++) {  
 str[i] = tolower((unsigned char)str[i]);  
 }  
}  
  
/\* 文字列配列のバブルソート \*/  
void bubble\_sort\_strings(char arr[][MAX\_LENGTH], int n)  
{  
 int i, j;  
 char temp[MAX\_LENGTH];  
   
 for (i = 0; i < n - 1; i++) {  
 for (j = 0; j < n - 1 - i; j++) {  
 if (strcmp(arr[j], arr[j + 1]) > 0) {  
 strcpy(temp, arr[j]);  
 strcpy(arr[j], arr[j + 1]);  
 strcpy(arr[j + 1], temp);  
 }  
 }  
 }  
}  
  
/\* 文字列配列での検索 \*/  
int search\_string(char arr[][MAX\_LENGTH], int count, char \*target)  
{  
 int i;  
   
 for (i = 0; i < count; i++) {  
 if (strcmp(arr[i], target) == 0) {  
 return i; /\* 見つかった場合のインデックス \*/  
 }  
 }  
 return -1; /\* 見つからない場合 \*/  
}  
  
int main(void)  
{  
 /\* 文字列の分割（トークン化） \*/  
 printf("=== 文字列の分割（トークン化） ===\n");  
 char input[] = "apple,banana,cherry,date,elderberry";  
 char tokens[MAX\_TOKENS][MAX\_LENGTH];  
 char temp[100];  
 char \*token;  
 int token\_count = 0;  
 int i;  
   
 /\* 入力文字列をコピー（strtokは元の文字列を変更するため） \*/  
 strcpy(temp, input);  
   
 /\* カンマで分割 \*/  
 token = strtok(temp, ",");  
 while (token != NULL && token\_count < MAX\_TOKENS) {  
 strcpy(tokens[token\_count], token);  
 token\_count++;  
 token = strtok(NULL, ",");  
 }  
   
 printf("分割結果:\n");  
 for (i = 0; i < token\_count; i++) {  
 printf("%d: %s\n", i + 1, tokens[i]);  
 }  
   
 /\* 文字列の変換（大文字・小文字） \*/  
 printf("\n=== 文字列の変換（大文字・小文字） ===\n");  
 char original[MAX\_STRINGS][MAX\_LENGTH] = {  
 "Hello World",  
 "Programming Language",  
 "C Language Tutorial",  
 "String Processing",  
 "Array Manipulation"  
 };  
 char uppercase\_copy[MAX\_STRINGS][MAX\_LENGTH];  
 char lowercase\_copy[MAX\_STRINGS][MAX\_LENGTH];  
   
 /\* 文字列をコピーして変換 \*/  
 for (i = 0; i < MAX\_STRINGS; i++) {  
 strcpy(uppercase\_copy[i], original[i]);  
 strcpy(lowercase\_copy[i], original[i]);  
   
 to\_uppercase(uppercase\_copy[i]);  
 to\_lowercase(lowercase\_copy[i]);  
 }  
   
 /\* 結果の表示 \*/  
 printf("元の文字列 -> 大文字 -> 小文字\n");  
 printf("=====================================\n");  
 for (i = 0; i < MAX\_STRINGS; i++) {  
 printf("%-20s -> %-20s -> %s\n",   
 original[i], uppercase\_copy[i], lowercase\_copy[i]);  
 }  
   
 /\* 文字列配列のソートと検索 \*/  
 printf("\n=== 文字列配列のソート ===\n");  
 char words[8][MAX\_LENGTH] = {  
 "Zebra", "Apple", "Monkey", "Banana",  
 "Cat", "Dog", "Elephant", "Fish"  
 };  
   
 printf("ソート前:\n");  
 for (i = 0; i < 8; i++) {  
 printf("%s ", words[i]);  
 }  
 printf("\n\n");  
   
 /\* バブルソート \*/  
 bubble\_sort\_strings(words, 8);  
   
 printf("ソート後:\n");  
 for (i = 0; i < 8; i++) {  
 printf("%s ", words[i]);  
 }  
 printf("\n");  
   
 /\* 検索のテスト \*/  
 printf("\n=== 文字列の検索 ===\n");  
 char cities[10][MAX\_LENGTH] = {  
 "Tokyo", "Osaka", "Kyoto", "Nagoya", "Sapporo",  
 "Fukuoka", "Kobe", "Sendai", "Hiroshima", "Yokohama"  
 };  
 char search\_target[] = "Kyoto";  
 int result;  
   
 printf("都市一覧:\n");  
 for (i = 0; i < 10; i++) {  
 printf("%d. %s\n", i + 1, cities[i]);  
 }  
   
 /\* 検索実行 \*/  
 result = search\_string(cities, 10, search\_target);  
 if (result != -1) {  
 printf("\n\"%s\" は %d番目にあります。\n", search\_target, result + 1);  
 } else {  
 printf("\n\"%s\" は見つかりませんでした。\n", search\_target);  
 }  
   
 /\* 簡易データベースの例 \*/  
 printf("\n=== 簡易従業員データベース ===\n");  
 char employee\_names[3][MAX\_LENGTH] = {  
 "田中太郎", "佐藤花子", "鈴木一郎"  
 };  
 char departments[3][MAX\_LENGTH] = {  
 "開発", "営業", "開発"  
 };  
 char positions[3][MAX\_LENGTH] = {  
 "課長", "部長", "一般"  
 };  
 int salaries[3] = {800000, 1200000, 500000};  
   
 /\* 全従業員の表示 \*/  
 printf("%-12s %-8s %-8s %s\n", "名前", "部署", "役職", "給与");  
 printf("--------------------------------------------------\n");  
 for (i = 0; i < 3; i++) {  
 printf("%-12s %-8s %-8s %d円\n",  
 employee\_names[i],  
 departments[i],  
 positions[i],  
 salaries[i]);  
 }  
   
 /\* 部署別集計 \*/  
 printf("\n=== 部署別従業員数 ===\n");  
 char unique\_depts[4][MAX\_LENGTH] = {"営業", "開発", "人事", "経理"};  
 int dept\_count;  
   
 for (i = 0; i < 4; i++) {  
 int count = 0;  
 int j;  
   
 for (j = 0; j < 3; j++) {  
 if (strcmp(departments[j], unique\_depts[i]) == 0) {  
 count++;  
 }  
 }  
 printf("%s: %d人\n", unique\_depts[i], count);  
 }  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 文字列の分割:  
 - strtok()関数による分割  
 - 元の文字列が変更されることに注意  
 - 区切り文字の指定方法  
  
2. 文字列変換:  
 - toupper()/tolower()による大文字小文字変換  
 - ctype.hライブラリの活用  
 - 文字列全体の変換処理  
  
3. 文字列ソート:  
 - バブルソートアルゴリズムの実装  
 - strcmp()による文字列比較  
 - 配列要素の交換処理  
  
4. 実用的な応用:  
 - データベース的な処理  
 - 集計・分析機能  
 - 検索・フィルタリング  
  
注意点:  
- strtok()は元の文字列を変更するため注意  
- 文字のキャストにはunsigned charを使用  
- バッファサイズの管理  
- NULL文字の適切な処理  
\*/

## string\_array\_operations.c

/\*  
 \* 文字列配列の実践例  
 \* ファイル名: string\_array\_operations.c  
 \* 説明: 二次元文字配列と文字列ポインタ配列の使い分けを学習  
 \*/  
  
#include <stdio.h>  
#include <string.h>  
  
#define MAX\_STRINGS 5  
#define MAX\_LENGTH 20  
  
int main(void)  
{  
 /\* 二次元文字配列での文字列配列 \*/  
 printf("=== 二次元文字配列による文字列配列 ===\n");  
 char languages[MAX\_STRINGS][MAX\_LENGTH];  
 int count = 0;  
 int i;  
   
 /\* 文字列配列への代入 \*/  
 strcpy(languages[count++], "C");  
 strcpy(languages[count++], "Python");  
 strcpy(languages[count++], "Java");  
 strcpy(languages[count++], "JavaScript");  
 strcpy(languages[count++], "C++");  
   
 /\* 表示 \*/  
 printf("プログラミング言語一覧:\n");  
 for (i = 0; i < count; i++) {  
 printf("%d. %s\n", i + 1, languages[i]);  
 }  
   
 /\* 文字列の変更 \*/  
 strcpy(languages[0], "C言語");  
 printf("\n変更後の1番目: %s\n", languages[0]);  
   
 /\* 文字列ポインタ配列での文字列配列 \*/  
 printf("\n=== 文字列ポインタ配列 ===\n");  
 char \*fruits[] = {  
 "Apple",  
 "Banana",   
 "Cherry",  
 "Date",  
 "Elderberry"  
 };  
 int fruit\_count = sizeof(fruits) / sizeof(fruits[0]);  
   
 /\* 表示 \*/  
 printf("果物一覧:\n");  
 for (i = 0; i < fruit\_count; i++) {  
 printf("%d. %s (長さ: %lu)\n", i + 1, fruits[i],   
 (unsigned long)strlen(fruits[i]));  
 }  
   
 /\* ポインタの変更（文字列リテラルの置き換え） \*/  
 fruits[0] = "リンゴ";  
 printf("\n変更後の1番目: %s\n", fruits[0]);  
   
 /\* 文字列配列の初期化と動的変更 \*/  
 printf("\n=== 動的に変更可能な文字列配列 ===\n");  
 char menu\_items[4][MAX\_LENGTH];  
 char \*categories[] = {"前菜", "メイン", "デザート", "飲み物"};  
 int item\_count = 0;  
   
 /\* メニュー項目の追加 \*/  
 strcpy(menu\_items[item\_count++], "サラダ");  
 strcpy(menu\_items[item\_count++], "ステーキ");  
 strcpy(menu\_items[item\_count++], "アイスクリーム");  
 strcpy(menu\_items[item\_count++], "コーヒー");  
   
 /\* カテゴリ別表示 \*/  
 printf("=== レストランメニュー ===\n");  
 for (i = 0; i < item\_count && i < 4; i++) {  
 printf("[%s] %s\n", categories[i], menu\_items[i]);  
 }  
   
 /\* 項目の変更 \*/  
 strcpy(menu\_items[1], "ハンバーグ");  
 printf("\nメイン料理を変更: %s\n", menu\_items[1]);  
   
 /\* メモリ使用量の比較 \*/  
 printf("\n=== メモリ使用量比較 ===\n");  
 char matrix\_strings[5][10] = {  
 "Cat", "Dog", "Bird", "Fish", "Rabbit"  
 };  
   
 char \*pointer\_strings[] = {  
 "Cat", "Dog", "Bird", "Fish", "Rabbit"  
 };  
   
 printf("二次元文字配列: %lu バイト\n",   
 (unsigned long)sizeof(matrix\_strings));  
 printf("文字列ポインタ配列: %lu バイト\n",   
 (unsigned long)sizeof(pointer\_strings));  
   
 printf("\n=== 実際の文字列長 ===\n");  
 int total\_chars = 0;  
 for (i = 0; i < 5; i++) {  
 int len = strlen(matrix\_strings[i]);  
 printf("%s: %d文字\n", matrix\_strings[i], len);  
 total\_chars += len;  
 }  
 printf("総文字数: %d文字\n", total\_chars);  
 printf("未使用領域: %lu バイト\n",   
 sizeof(matrix\_strings) - total\_chars - 5);  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 二次元文字配列:  
 - 固定サイズのメモリ確保  
 - 文字列の変更が可能  
 - メモリ使用効率は低い場合がある  
   
2. 文字列ポインタ配列:  
 - ポインタのみを格納  
 - 文字列リテラルは変更不可  
 - メモリ使用効率が良い  
   
3. 使い分けの基準:  
 - 文字列を変更する必要がある: 二次元配列  
 - 文字列が固定: ポインタ配列  
 - メモリ効率を重視: ポインタ配列  
   
4. 実用的な応用:  
 - メニュー管理システム  
 - 設定項目の管理  
 - データの分類・整理  
  
注意点:  
- 文字列リテラルの変更は未定義動作  
- バッファサイズの確認  
- 配列の境界チェック  
\*/

## string\_basic.c

/\*  
 \* 文字列操作の基本例  
 \* ファイル名: string\_basic.c  
 \* 説明: 標準ライブラリの文字列操作関数の使用方法を学習  
 \*/  
  
#include <stdio.h>  
#include <string.h>  
  
int main(void)  
{  
 /\* 文字列の表現方法 \*/  
 char str1[] = {'H', 'e', 'l', 'l', 'o', '\0'};  
 char str2[] = "Hello";  
 char str3[10] = "Hello";  
 char \*str4 = "Hello";  
   
 printf("=== 文字列の表現方法 ===\n");  
 printf("str1: %s\n", str1);  
 printf("str2: %s\n", str2);  
 printf("str3: %s\n", str3);  
 printf("str4: %s\n", str4);  
   
 /\* 文字列の長さとサイズ \*/  
 printf("\n=== 文字列の長さとサイズ ===\n");  
 printf("文字列: \"%s\"\n", str2);  
 printf("strlen(str2): %lu文字\n", (unsigned long)strlen(str2));  
 printf("sizeof(str2): %lu バイト\n", (unsigned long)sizeof(str2));  
   
 /\* strcpy() - 文字列のコピー \*/  
 printf("\n=== 文字列のコピー ===\n");  
 char source[] = "Hello, World!";  
 char destination[50];  
   
 strcpy(destination, source);  
 printf("destination: %s\n", destination);  
   
 /\* 部分的なコピー \*/  
 strncpy(destination, source, 5);  
 destination[5] = '\0'; /\* null終端を明示的に追加 \*/  
 printf("partial copy: %s\n", destination); /\* "Hello" \*/  
   
 /\* strcat() - 文字列の連結 \*/  
 printf("\n=== 文字列の連結 ===\n");  
 char str\_a[50] = "Hello";  
 char str\_b[] = ", World!";  
 char str\_c[] = " How are you?";  
   
 strcat(str\_a, str\_b);  
 printf("After strcat: %s\n", str\_a); /\* "Hello, World!" \*/  
   
 /\* 部分的な連結 \*/  
 strncat(str\_a, str\_c, 4);  
 printf("After strncat: %s\n", str\_a); /\* "Hello, World! How" \*/  
   
 /\* strcmp() - 文字列の比較 \*/  
 printf("\n=== 文字列の比較 ===\n");  
 char apple[] = "Apple";  
 char banana[] = "Banana";  
 char apple2[] = "Apple";  
 int result;  
   
 result = strcmp(apple, banana);  
 if (result < 0) {  
 printf("\"%s\" < \"%s\"\n", apple, banana);  
 } else if (result > 0) {  
 printf("\"%s\" > \"%s\"\n", apple, banana);  
 } else {  
 printf("\"%s\" == \"%s\"\n", apple, banana);  
 }  
   
 /\* 等価性の確認 \*/  
 if (strcmp(apple, apple2) == 0) {  
 printf("\"%s\" と \"%s\" は同じです\n", apple, apple2);  
 }  
   
 /\* 部分比較 \*/  
 if (strncmp(apple, banana, 1) != 0) {  
 printf("最初の文字が異なります\n");  
 }  
   
 /\* 文字列検索関数 \*/  
 printf("\n=== 文字列検索関数 ===\n");  
 char text[] = "Hello, World! Welcome to C programming.";  
 char \*found;  
   
 /\* 文字の検索 \*/  
 found = strchr(text, 'W');  
 if (found != NULL) {  
 printf("'W' found at position: %ld\n", found - text);  
 }  
   
 /\* 文字列の検索 \*/  
 found = strstr(text, "World");  
 if (found != NULL) {  
 printf("\"World\" found at position: %ld\n", found - text);  
 printf("Found: %s\n", found);  
 }  
   
 /\* 最後の文字を検索 \*/  
 found = strrchr(text, 'o');  
 if (found != NULL) {  
 printf("Last 'o' at position: %ld\n", found - text);  
 }  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 文字列の表現方法:  
 - 文字配列による初期化  
 - 文字列リテラルによる初期化  
 - ポインタによる文字列参照  
  
2. 標準ライブラリ関数:  
 - strlen(): 文字列の長さを取得  
 - strcpy()/strncpy(): 文字列のコピー  
 - strcat()/strncat(): 文字列の連結  
 - strcmp()/strncmp(): 文字列の比較  
  
3. 文字列検索:  
 - strchr(): 文字の検索  
 - strrchr(): 文字の後方検索  
 - strstr(): 部分文字列の検索  
  
注意点:  
- strncpy()使用時はnull終端文字の確認が必要  
- バッファオーバーフローの防止  
- NULLポインタのチェック  
\*/