C/C++プログラミング

第8回

文字列とポインタの演習

文字列とポインタ: C++

List8-1 文字列リテラル

```
// 文字列リテラルの型と大きさを表示

#include <iostream>
#include <typeinfo>

using namespace std;

int main()
{
        cout << "■文字列リテラル\"ABC\"\n";
        cout << "  型："     << typeid("ABC").name()
             << "  大きさ：" << sizeof("ABC") << "\n\n";

        cout << "■文字列リテラル\"\"\n";
        cout << "  型："     << typeid("").name()
             << "  大きさ：" << sizeof("") << "\n\n";

        cout << "■文字列リテラル\"ABC\\0DEF\"\n";
        cout << "  型："     << typeid("ABC\0DEF").name()
             << "  大きさ：" << sizeof("ABC\0DEF") << "\n";
}
```

文字列とポインタ: C++

「
```
cout << "■文字列リテラル\"ABC\"\n";
cout << "  型："    << typeid("ABC").name()
     << "  大きさ：" << sizeof("ABC") << "\n\n";
```
」

二重引用符「"」で囲んだ文字列を文字列リテラルという.
文字列リテラルは計算機のメモリ上に const char 型 の配列として格納され，文字列の最後に「ナル文字」である「\0」が自動付加される．例えば "ABC" はメモリ上は「ABC\0」と格納される．より具体的には A, B, C, \0 の合計4つの要素からなるchar型の配列として格納される.

「 typeid("ABC").name() 」によって，文字列リテラル「ABC\0」がメモリ上に確保され，かつその「型」の「名前」を表す．この例の場合要素数4のchar型配列である.
実行結果は教科書では「char const [4]」となっているが，コンパイラに依存する.

「 sizeof("ABC") 」によって文字列リテラル「ABC\0」の配列の要素数4を表す.

「cout << " 型："   << typeid("").name()」では文字列リテラル「""」＝「"\0"」の場合の型の名前が表示される．要素数は1である.

「cout << " 型："   << typeid("ABC\0DEF").name()」では文字列リテラル「"ABC\0DEF"」=「"ABC\0DEF\0"」の場合の型の名前が表示される．要素数は8．ただし「ナル文字」は文字列の最後を表すので，これは文字列「ABC」と文字列「DEF」が連なった文字列リテラルとみなされる.

文字列とポインタ: C++

Fig.8-1 文字列リテラルとその内部

文字列とポインタ: C++

List8-2 文字列リテラルと配列

```
----------------------------------------
// 配列に文字列を格納して表示（代入）

#include <iostream>

using namespace std;

int main()
{
        char s[4];          // 文字列を格納する配列

        s[0] = 'A';         // 代入
        s[1] = 'B';         // 代入
        s[2] = 'C';         // 代入
        s[3] = '\0'; // 代入

        cout << "配列sに入っている文字列は\"" << s << "\"です。\n";// 表示
}
----------------------------------------
```

文字列とポインタ: C++

「
```
char s[4];
```
」
文字列リテラルはメモリ上ではchar型配列として格納されるのであった．そこで文字列を格納するための領域を確保するために char s[4] と宣言する．

「
```
s[0] = 'A';                // 代入
s[1] = 'B';                // 代入
s[2] = 'C';                // 代入
s[3] = '\0'; // 代入
```
」
配列 s[4] の各要素に文字 'A', 'B', 'C', '\0' を格納していく．

「
```
cout << "配列sに入っている文字列は\"" << s << "\"です。\n";
```
」
配列 s[4] に対して「s」はその配列の先頭のアドレスを表すのだった．
cout << s とすると，その配列の各要素が順番に出力される．この例の場合，「ABC」と表示される．

文字列とポインタ: C++

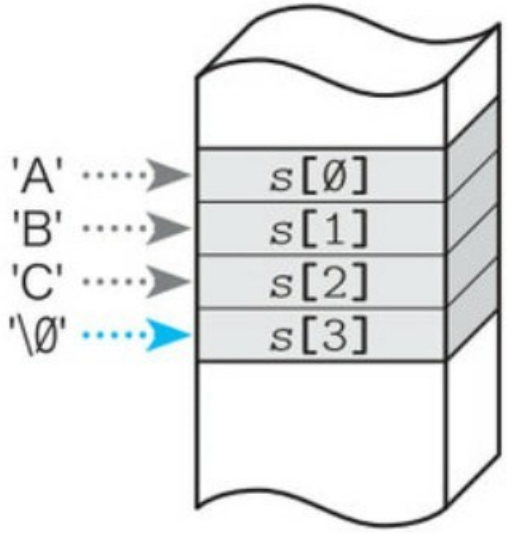


Fig.8-2　配列に格納する文字列

文字列とポインタ: C++

List8-3 文字列リテラルと配列

```
----------------------------------------
// 配列に文字列を格納して表示（初期化）

#include <iostream>

using namespace std;

int main()
{
        char s1[] = {'A', 'B', 'C', '\0'};
        char s2[] = {"ABC"};
        char s3[] = "ABC";

        cout << "配列s1に文字列\"" << s1 << "\"が格納されています。\n";
        cout << "配列s2に文字列\"" << s2 << "\"が格納されています。\n";
        cout << "配列s3に文字列\"" << s3 << "\"が格納されています。\n";
}
----------------------------------------
```

文字列とポインタ: C++

「

```
char s1[] = {'A', 'B', 'C', '\0'};
char s2[] = {"ABC"};
char s3[] = "ABC";
```

」

文字列配列の様々な初期化．結果は同じになる．

文字列とポインタ: C++

List8-5 キーボードからの文字列の読み込み

```
----------------------------------------
// 名前を尋ねて挨拶（文字列の読込みと表示）

#include <iostream>

using namespace std;

int main()
{
        char name[36];

        cout << "お名前は：";
        cin >> name;

        cout << "こんにちは、" << name << "さん!!\n";
}
----------------------------------------
```

文字列とポインタ: C++

「

```
char name[36];

cout << "お名前は：";
cin >> name;
```

」

文字列配列にキーボードから入力．この例の場合，36文字以上の入力は保証されない．

読み込む文字数に制限を設けたい場合は，プログラムの網かけ部を以下のコードに置きかえます

「

```
char name[36];

cout << "お名前は：";
cin.getline(name, 36);
```

」

文字列とポインタ: C++

List8-6 関数間の文字列の受け渡し

```
----------------------------------------
// 受け取った文字列を表示

#include <cctype>
#include <iostream>

using namespace std;

//--- 文字列sを表示 ---//
void put_str(const char s[])
{
        for (int i = 0; s[i] != 0; i++)
                cout << s[i];
}

int main()
{
        char str[36];

        put_str("文字列 : ");
        cin.getline(str, 36);

        put_str(str);
        cout << '\n';
}
----------------------------------------
```

文字列とポインタ: C++

「
```
void put_str(const char s[])
{
    for (int i = 0; s[i] != 0; i++)
    cout << s[i];
}
```
」
文字列を表示する関数．この関数を呼び出す関数（本例の場合はmain関数）から文字列リテラルのアドレスあるいは文字列配列のアドレスが「const char s[]」に引き渡されコピーされる．この関数の本体では文字列配列 s[ ] の各要素 s[i] を，s[i] の中身がナル文字でない限り（つまりs[i] != 0）標準出力に出力する．

「
```
put_str("文字列：");
```
」
文字列リテラル "文字列：" を put_str に引き渡し画面表示させる．

「
```
char str[36];
cin >> str;
put_str(str);
```
」
文字列配列 str を定義し，str に文字列をキーボード入力する．put_str(str) では文字列配列の先頭アドレスを　put_str に引き渡し，画面印字させる．

文字列とポインタ: C++

List8-8 ポインタによる文字列の扱い

```
---------------------------------------
// 配列による文字列とポインタによる文字列

#include <iostream>

using namespace std;

int main()
{
        char  str[] = "ABC";                    // 配列による文字列
        char* ptr   = "123";                    // ポインタによる文字列

        cout << "str = \"" << str << "\"\n";
        cout << "ptr = \"" << ptr << "\"\n";
}
---------------------------------------
```

文字列とポインタ: C++

「
```
char* ptr   = "123";

cout << "ptr = \"" << ptr << "\"\n";
```
」
ポインタの章で説明したように，ポインタを用いて配列を扱えた．そこでこの例では，
ポインタ変数 ptr を文字列リテラル "123" を代入することにより，対応する文字列配列の先頭アドレスを ptr にセットする．
これ以降，ptr は"123"が格納された文字列配列の先頭アドレスを表すので，
「cout << ptr」で "123" の文字列が標準出力に表示される．

注意：危険な書き方
```
char* ptr = "123";
```

上記の "123" は文字列リテラル（定数）であり，実体は const char[4] ですこれを char* に代入すると const が外れ，*ptr = 'X' などで書き換えを行った場合，未定義動作（典型的にはクラッシュやメモリエラー）になります。

変更しない前提の推奨記法（安全）
```
const char str[] = "123";   // 書き換え不可の配列
```

文字列とポインタ: C++

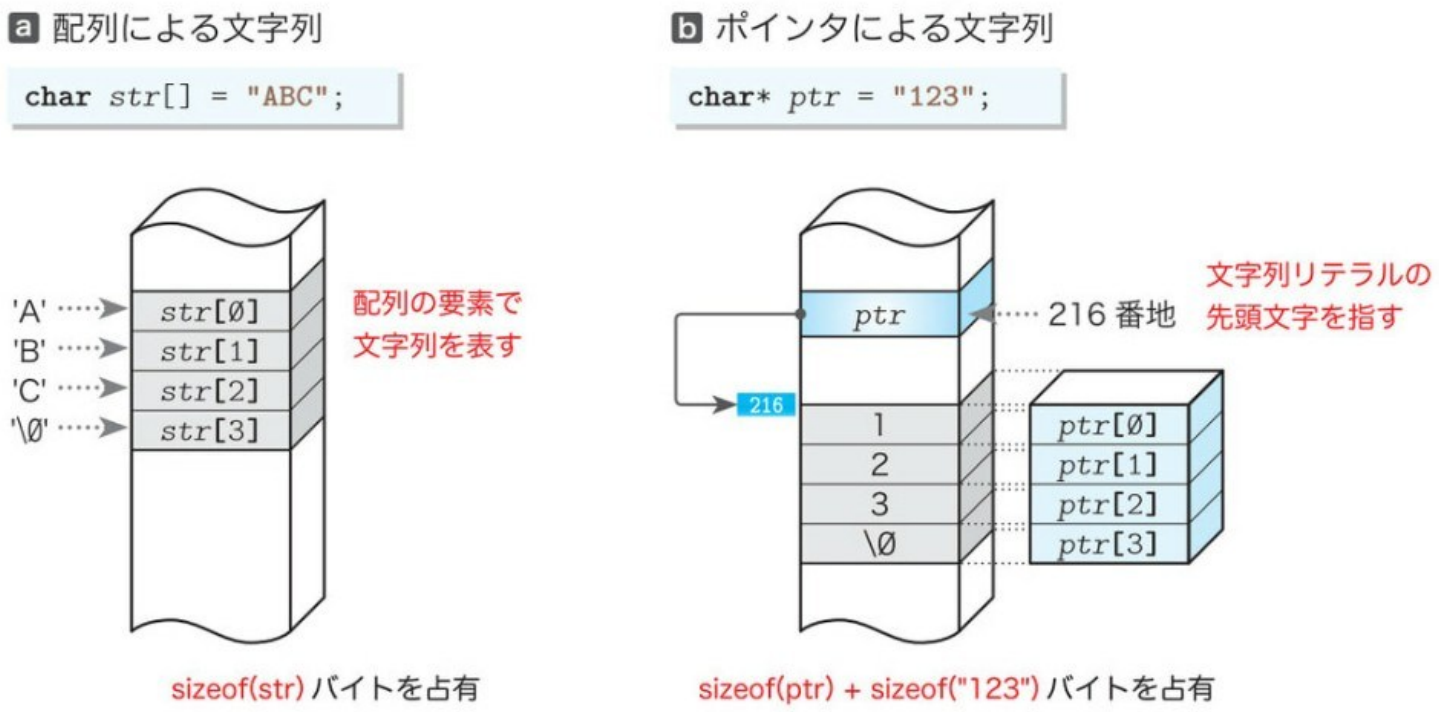


Fig.8-5 配列による文字列とポインタによる文字列

文字列とポインタ: C++

List8-11 文字列の配列

```
----------------------------------------
// 配列による文字列とポインタによる文字列

#include <iostream>

using namespace std;

int main()
{
    char a[][5] = {"LISP", "C", "Ada"};     // 配列による文字列の配列
    char* p[]   = {"PAUL", "X", "MAC"};     // ポインタによる文字列の配列

    for (int i = 0; i < 3; i++)
        cout << (void *)a[i] << ", " << "a[" << i << "] = \"" << a[i] << "\"\n";

    for (int i = 0; i < 3; i++)
        cout << (void *)p[i] << ", " << "p[" << i << "] = \"" << p[i] << "\"\n";
}
----------------------------------------
```

文字列とポインタ: C++

「
```
char a[][5] = {"LISP", "C", "Ada"};     // 配列による文字列の配列
char* p[]   = {"PAUL", "X", "MAC"};     // ポインタによる文字列の配列
```
」
1行目は「要素型が要素数5のchar型配列」の配列を宣言し，a[0][] は文字列リテラル "LISP"，a[1][] は "C"，a[2][] は "Ada"，と初期化する．
2行目はchar型のポインタの配列を宣言し，p[0] は文字列リテラル "PAUL" へのポインタ，p[1] は "X" へのポインタ，p[2] は "MAC" へのポインタ，と初期化する．

「
```
for (int i = 0; i < 3; i++)
    cout << (void *)a[i] << ", " << "a[" << i << "] = \"" << a[i] << "\"\n";
```
」
「
```
  for (int i = 0; i < 3; i++)
      cout << (void *)p[i] << ", " << "p[" << i << "] = \"" << p[i] << "\"\n";
```
」
それぞれ多次元配列 a，ポインタ配列 p を用いて文字列を印字する．

文字列とポインタ: C++

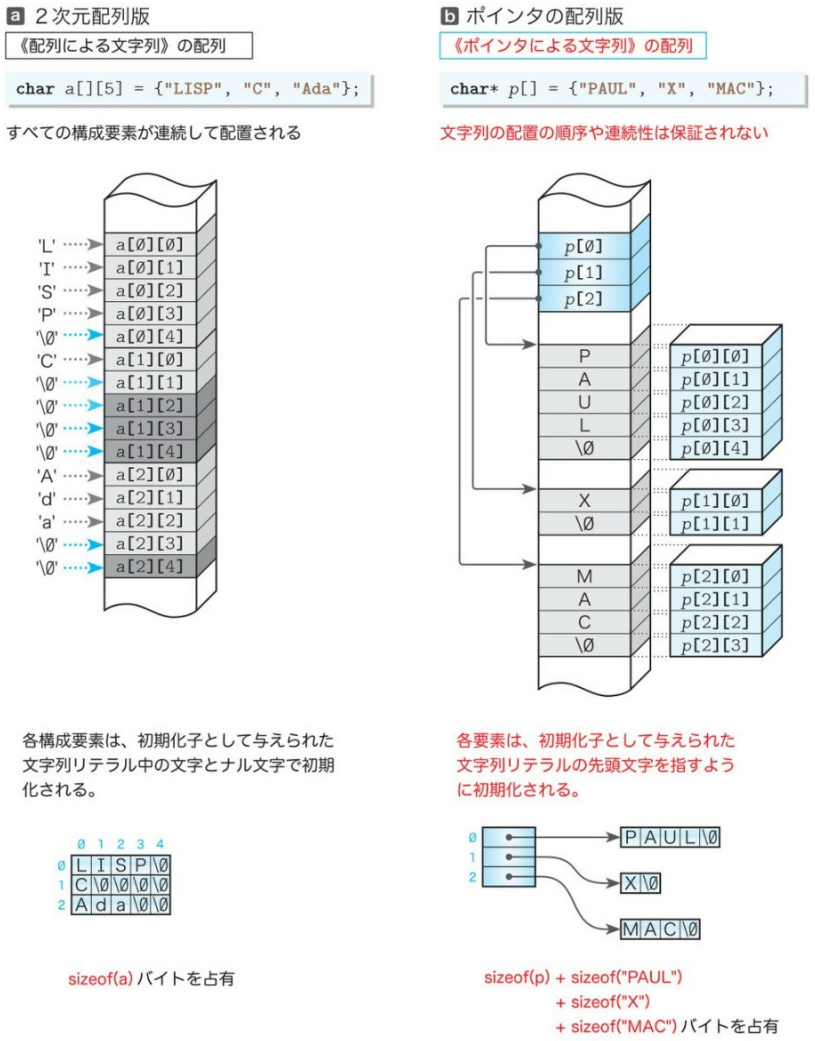


Fig.8-7 文字列の配列（2次元配列とポインタの配列）

cstring ライブラリ: C++

List8-14 strlen 関数

```
---------------------------------------
// strlen関数の利用例（その１）

#include <cstring>
#include <iostream>

using namespace std;

int main(void)
{
    char str[100];

    cout << "文字列を入力してください : ";
    cin >> str;

    cout << "文字列\"" << str << "\"の長さは" << strlen(str) << "です。\n";
}
---------------------------------------
```

cstring ライブラリ: C++

「
```
#include <cstring>
```
」
cstring ライブラリを用いるためのヘッダ

「
```
cout << "文字列\"" << str << "\"の長さは" << strlen(str) << "です。\n";
```
」
関数 strlen は文字列の長さを返す.

「日本語」と入力した場合、端末が UTF-8 エンコーディングを使用していると、一つの漢字は通常 3バイトになります。

cstring ライブラリ: C++

strcpy, strncpy：文字列をコピーする

文字列をコピーするのが、**strcpy 関数**と**strncpy 関数**です。**strcpy** 関数は、文字列を丸ごとコピーします。もう一つの **strncpy** 関数は、文字数に制限を設けた上でコピーを行う関数です（**Fig.8-9**）。

	strcpy
ヘッダ	`#include <cstring>`
形　式	`char* strcpy(char* s1, const char* s2);`
解　説	*s2* が指す文字列を、*s1* が指す配列にコピーする。コピー元とコピー先が重なる場合の動作は定義されない。
返却値	*s1* の値を返す。

	strncpy
ヘッダ	`#include <cstring>`
形　式	`char* strncpy(char* s1, const char* s2, size_t n);`
解　説	*s2* が指す文字列を、*s1* が指す配列にコピーする。*s2* の長さが *n* 以上の場合は *n* 文字までをコピーし、*n* より短い場合は残りをナル文字で埋めつくす。コピー元とコピー先が重なる場合の動作は定義されない。
返却値	*s1* の値を返す。

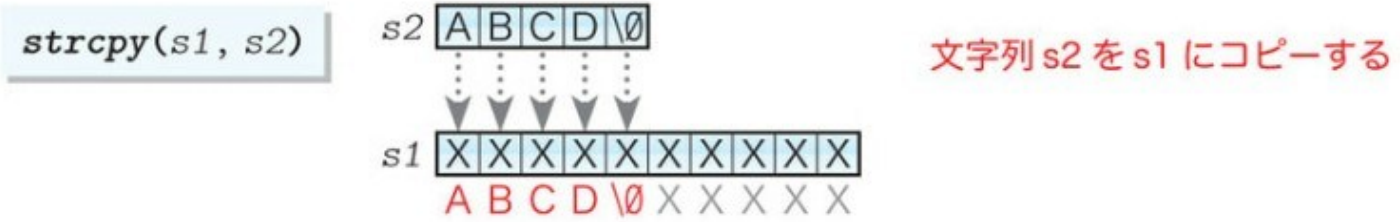
cstring ライブラリ: C++

List8-16 strcpy 関数と strncpy 関数

```
// strcpy関数とstrncpy関数の利用例

#include <cstring>
#include <iostream>

using namespace std;

int main(void)
{
    char tmp[16];
    char s1[16], s2[16], s3[16];

    cout << "文字列を入力してください：";
    cin >> tmp;

    strcpy(s1, strcpy(s2, tmp));    // s2にコピーした文字列をs1にもコピー

    cout << "文字列s1は\"" << s1 << "\"です。\n";
    cout << "文字列s2は\"" << s2 << "\"です。\n";
    cout << "文字列s3は\"" << strcpy(s3, tmp) << "\"です。\n";

    char* x = "XXXXXXXXX";          // 9個の'X'とナル文字 */

    strcpy(s3, x); strncpy(s3, "12345", 3);      cout << "s3 = " << s3 << '\n';
    strcpy(s3, x); strncpy(s3, "12345", 5);      cout << "s3 = " << s3 << '\n';
    strcpy(s3, x); strncpy(s3, "12345", 7);      cout << "s3 = " << s3 << '\n';
    strcpy(s3, x); strncpy(s3, "1234567890", 9); cout << "s3 = " << s3 << '\n';
}
```

cstring ライブラリ: C++

「
strcpy(s1, strcpy(s2, tmp));　// s2にコピーした文字列をs1にもコピー
」
strcpy(s2, tmp) は tmp の文字列を s2 にコピーする．返値は s2 .
strcpy(s1, strcpy(s2, tmp)) は s2 の文字列を s1 にコピーする．返値は s1.
コピーのルールは下の図の通り.



cstring ライブラリ: C++

「
```
strncpy(s3, "12345", 3);
strncpy(s3, "12345", 5);
strncpy(s3, "12345", 7);
strncpy(s3, "1234567890", 9);
```
」

strncpy(s3, "12345", 3) は文字列 "12345" の初めの3文字（"123"）をs3の文字列にコピーする．返値はs3．以下同様．コピーのルールは下の図．

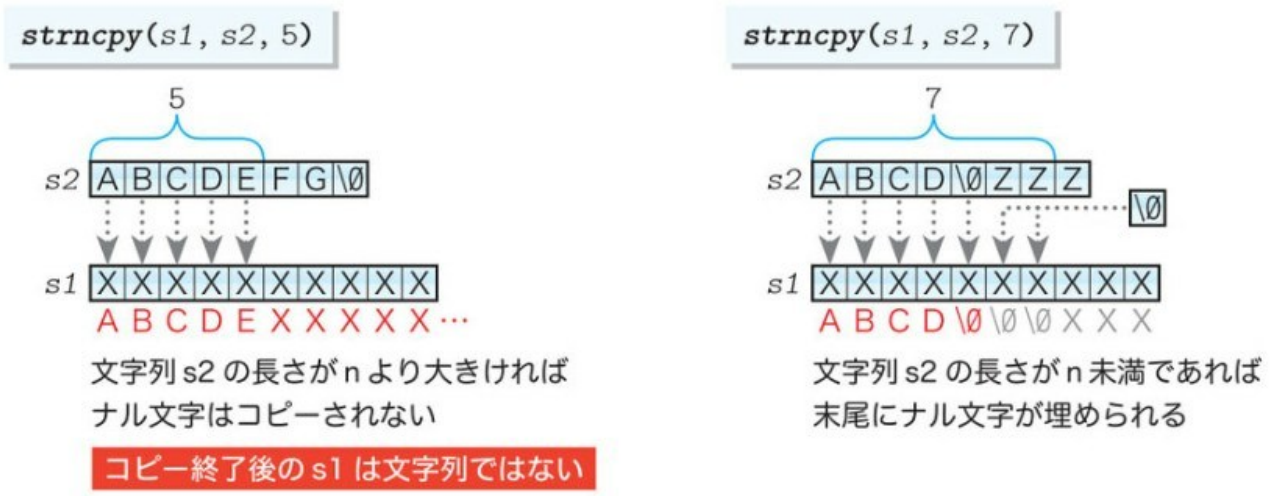


Fig.8-9 strcpy 関数と strncpy 関数の働き

cstring ライブラリ: C++

strcat, strncat：文字列を連結する

strcat 関数と**strncat 関数**は、文字列を連結する関数です。前者は文字列を丸ごと連結し、後者は文字数に制限を設けた上での連結を行います。

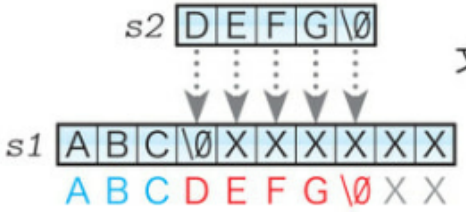
▶ “cat” は、『連結する』という意味の concatenate に由来します（猫ではありません）。

	strcat
ヘッダ	`#include <cstring>`
形　式	`char* strcat(char* s1, const char* s2);`
解　説	*s2* が指す文字列を、*s1* が指す文字列の末尾に連結する。コピー元とコピー先が重なる場合の動作は定義されない。
返却値	*s1* の値を返す。

	strncat
ヘッダ	`#include <cstring>`
形　式	`char* strncat(char* s1, const char* s2, size_t n)`
解　説	*s2* が指す文字列を、*s1* が指す文字列の末尾に連結する。*s2* の長さが *n* より長い場合は、切り捨てる。コピー元とコピー先が重なる場合の動作は定義されない。
返却値	*s1* の値を返す。

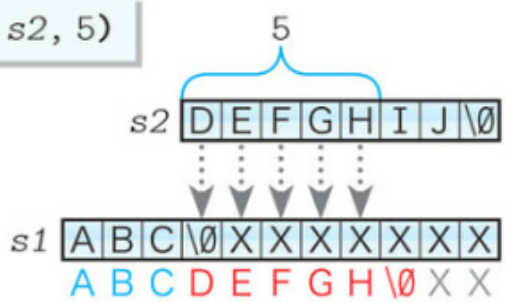
cstring ライブラリ: C++

strcat(*s1*, *s2*)



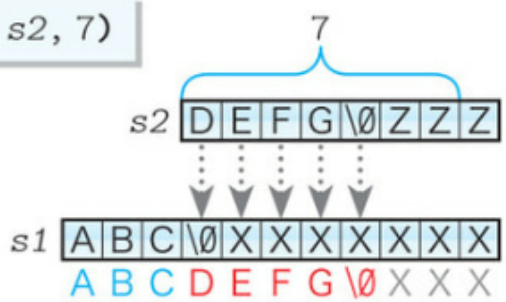
文字列 s2 を s1 の末尾に連結する

strncat(*s1*, *s2*, 5)



文字列 s2 の長さが n より大きければ
s2 の先頭 n 文字とナル文字を連結

strncat(*s1*, *s2*, 7)



文字列 s2 の長さが n 未満であれば
ナル文字までを連結する

Fig.8-11 strcat 関数と strncat 関数の働き

cstring ライブラリ: C++

List8-17 strcat 関数と strncat 関数

```
----------------------------------------
// strcat関数とstrncat関数の利用例

#include <cstring>
#include <iostream>

using namespace std;

int main(void)
{
    char s[10];
    char* x = "ABC";

    strcpy(s, "QWE");   // sは"QWE"となる
    strcat(s, "RTY");   // sは"QWERTY"となる
    cout << "s = " << s << '\n';

    strcpy(s, x);   strncat(s, "123", 1);       cout << "s = " << s << '\n';
    strcpy(s, x);   strncat(s, "123", 3);       cout << "s = " << s << '\n';
    strcpy(s, x);   strncat(s, "123", 5);       cout << "s = " << s << '\n';
    strcpy(s, x);   strncat(s, "12345", 5);     cout << "s = " << s << '\n';
    strcpy(s, x);   strncat(s, "123456789", 5); cout << "s = " << s << '\n';
}
----------------------------------------
```

cstring ライブラリ: C++

strcmp, strncmp：文字列を比較する

strcmp 関数と**strncmp 関数**は、二つの文字列を比較する関数です。文字列を丸ごと比較するのが**strcmp**関数で、比較する文字数に制限を設けるのが**strncmp**関数です。

strcmp	
ヘッダ	`#include <cstring>`
形　式	`int strcmp(const char* s1, const char* s2);`
解　説	*s1* が指す文字列と *s2* が指す文字列の大小関係（先頭から順に 1 文字ずつ **unsigned char** 型の値として比較していき、異なる文字が出現したときに、それらの文字の対に成立する大小関係とする）の比較を行う。
返却値	等しければ 0、*s1* が *s2* より大きければ正の整数値、*s1* が *s2* より小さければ負の整数値を返す。

strncmp	
ヘッダ	`#include <cstring>`
形　式	`int strncmp(const char* s1, const char* s2, size_t n);`
解　説	*s1* が指す文字の配列と *s2* が指す文字の配列の先頭 *n* 文字までの大小関係の比較を行う。ナル文字以降の文字は比較しない。
返却値	等しければ 0、*s1* が *s2* より大きければ正の整数値、*s1* が *s2* より小さければ負の整数値を返す。

cstring ライブラリ: C++

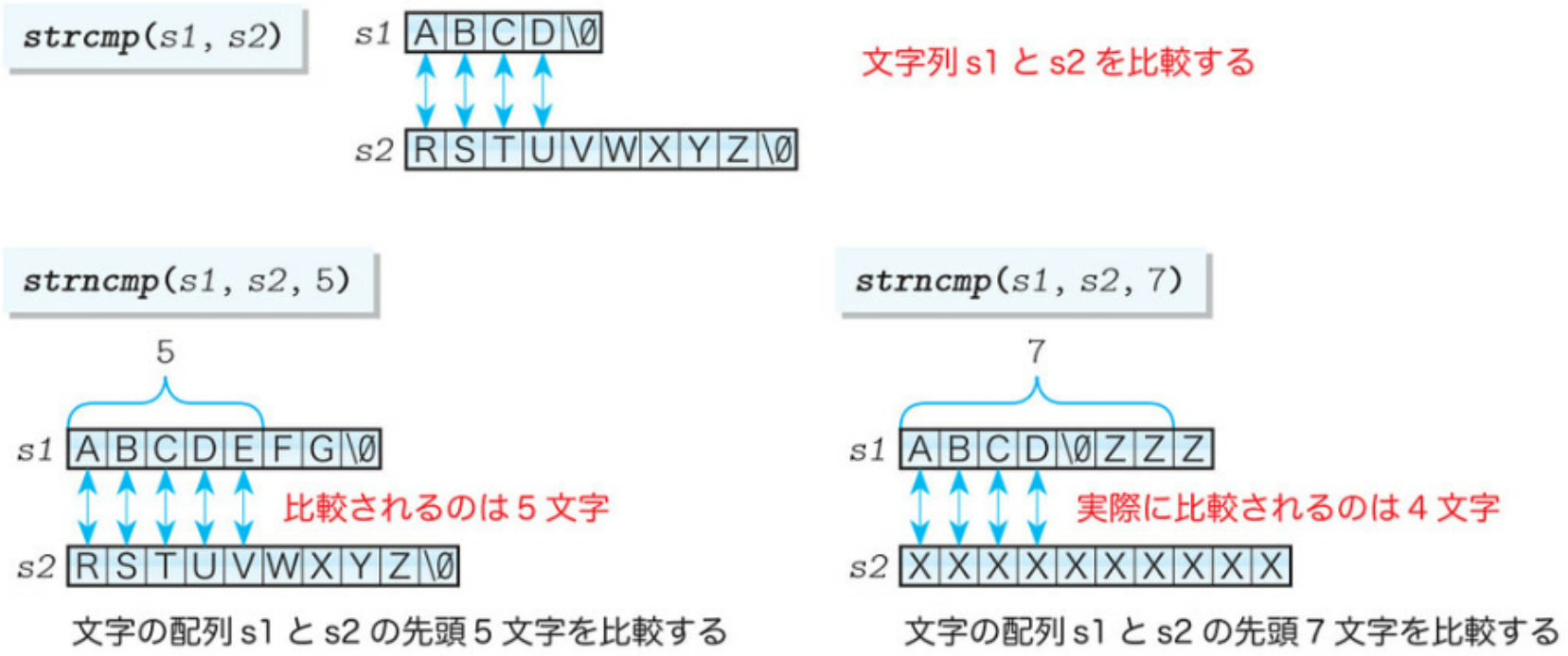


Fig.8-12 strcmp 関数と strncmp 関数の働き

cstring ライブラリ: C++

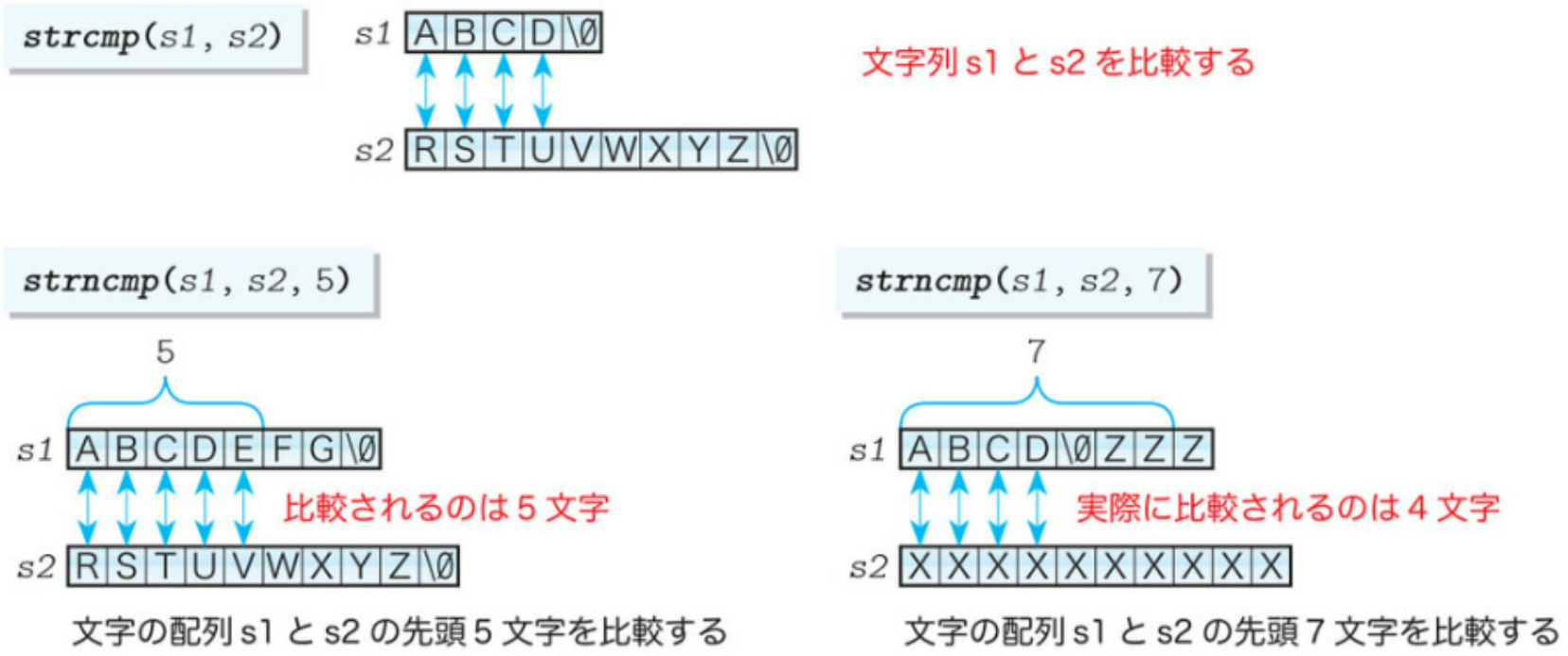


Fig.8-12 strcmp 関数と strncmp 関数の働き

cstring ライブラリ: C++

本日の授業内容は、C/C++ における**文字列のメモリ割り当てと利用様式**の理解を目的とします。

- C言語では、char str[len]; で定義される「文字列」の本質は**文字配列**であり、**末尾に '\0' が必要**です。
- C言語は初期の高級言語として、**メモリと末尾の '\0' を開発者自身が注意深く管理**する必要があります。
- 入力・連結・コピー等の操作では**必ず長さを明示的に制限**してください。**境界を超えると未定義動作**となり、不具合は潜伏的で深刻、かつ原因追跡が困難です。

実務では、**std::string を優先的に使用**してください。十分にカプセル化された文字列型であり、**より安全で可読性が高く、扱いやすい**です。

cstring ライブラリ: C++

std::string のよく使う操作

```
// 1) 文字列の構築と代入
string s  = "abc";      // 直接初期化
string s2 = "123456";
s2 = s;                 // 代入（コピー）

// 2) 結合（連結）
string s3 = s + s2;     // 演算子 +
s2 = s2 + "!@#";        // さらに連結

// 3) アクセスと属性
cout << "s.size()=" << s.size() << ", s.empty()=" << s.empty() << "\n";
cout << "s[0]=" << s[0] << ", s.at(1)=" << s.at(1) << "\n"; // at は境界チェックあり

// 4) 比較（辞書順）
if (s == s2)      cout << "s == s2\n";
else if (s < s2)  cout << "s < s2\n";
else              cout << "s > s2\n";

// 5) 部分文字列：substr(pos, len)
//   len を省略すると末尾まで。pos > size の場合は例外を投げる
string sub1 = s3.substr(1, 3);  // 位置1から3文字
string sub2 = s3.substr(2);     // 位置2から末尾まで
cout << "sub1=" << sub1 << ", sub2=" << sub2 << "\n";

// 6) 入出力（cin / cout）
cout << "単語を入力してください: ";
cin >> s;             //   単語入力：空白で区切られる
cout << "入力(単語) = " << s << "\n";
```

課題

課題8-1 （**プログラム名：ex08_1.cpp**）

文字列を逆順に並べ替える関数を実装せよ。

- 関数プロトタイプ
  void reverse_string(char str[]);
- main で長さ10未満の文字列（ 英数字のみ ）をキーボードから読み取り、reverse_string を呼び出してその場で逆順に並べ替え、結果を出力すること。

【実行例】

例1
入力：
abcdef⏎

出力：
fedcba

例2
入力：
a⏎

出力：
a⏎

⏎ : enter

```
#include <iostream>
#include <cstring>
using namespace std;

void reverse_string(char str[])
{
    // この関数を完成させること
}


int main()
{
    char s[10] = {0};
    cin.getline(s, 10);
    reverse_string(s);
    cout << s << '\n';
}
```

課題

課題8-2 （**プログラム名：ex08_2.cpp**）

2つの文字列を読み取り「A and B」の形式で出力せよ。

- キーボードから 6文字以内 の文字列（**日本語および英数字の入力を可**）を2つ読み取りなさい。
- 出力は <1つ目> and <2つ目>（両側に半角スペースを入れる）としなさい。

【実行例】

例1
入力：
abc⏎
123⏎

出力：
abc and 123

例2
入力：
麗澤大学⏎
世界⏎

出力：
麗澤大学 and 世界

⏎ : enter

課題

課題8-3 （**プログラム名：ex08_3.cpp**）

部分文字列判定

- キーボードから文字列 A と B（長さ10未満、英数字のみ ） を読み取り，B が A の部分文字列であるかを判定しなさい。
- 部分文字列であれば 「True」，そうでなければ 「False」を出力しなさい。

【実行例】

例1
入力：
abcdef⏎
cd⏎

出力：
True

例2
入力：
abc⏎
abc⏎

出力：
True

例3
入力：
cd⏎
abcdef⏎

出力：
False

例4
入力：
aabbcc⏎
abc⏎

出力：
False

⏎ : enter